The University of Nottingham

UNITED KINGDOM · CHINA · MALAYSIA

School of Computer Science

Heuristic Decomposition and Mathematical Programming for Workforce Scheduling and Routing Problems

Wasakorn Laesanklang

Thesis submitted to the University of Nottingham for the degree of Doctor of Philosophy

January 2017

Abstract

This thesis presents a PhD research project using a mathematical programming approach to solve a home healthcare problem (HHC) as well as general workforce scheduling and routing problems (WSRPs). In general, the workforce scheduling and routing problem consists of producing a schedule for mobile workers to make visits at different locations in order to perform some tasks. In some cases, visits may have time-wise dependencies in which a visit must be made within a time period depending on the other visit. A home healthcare problem is a variant of workforce scheduling and routing problems, which consists of producing a daily schedule for nurses or care workers to visit patients at their home. The scheduler must select qualified workers to make visits and route them throughout the time horizon.

We implement a mixed integer programming model to solve the HHC. The model is an adaptation of the WSRP from the literature. However, the MIP solver cannot solve a large-scale real-world problem defined in this model form because the problem requires large amounts of memory and computational time. To tackle the problem, we propose heuristic decomposition approaches which split a main problem into sub-problems heuristically and each sub-problem is solved to optimality by the MIP solver. The first decomposition approach is a geographical decomposition with conflict avoidance (GDCA). The algorithm avoids conflicting assignments by solving sub-problems in a sequence in which worker's availabilities are updated after a sub-problem is solved. The approach can find a feasible solution for every HHC problem instance tackled in this thesis. The second approach is a decomposition with conflict repair and we propose two variants: geographical decomposition with conflict repair (GDCR) and repeated decomposition and conflict repair (RDCR). The GDCR works in the same way as GDCA but instead of solving sub-problems in a given sequence, they are solved with no specific order and conflicting assignments are allowed. Later on, the conflicting assignments are resolved by a conflicting assignments repair process. The remaining unassigned visits are allocated by a heuristic assignment algorithm. The second variant, RDCR, tackles the unassigned visits by repeating the decomposition and conflict repair until no further improvement has been found. We also conduct an experiment to use different decomposition rules for RDCR. Based on computational experiments conducted in this thesis, the RDCR is found to be the best of the heuristic decomposition approaches. Therefore, the RDCR is extended to solve a WSRP with time-dependent activities constraints. The approach requires modification to accommodate the time-dependent activities constraints which means that two visits may have time-wise requirements such as synchronisation, time overlapped, etc.

In addition, we propose a reformulated MIP model to solve the HHC problem. The new model is considered to be a compact model because it has significantly fewer constraints. The aim of the reformulation is to reduce the solver requirements for memory and computational time. The MIP solver can solve all the HHC instances formulated in a compact model. Most of solutions obtained with this approach are the best known solutions so far except for those the instances for which the optimal solution can be found using the full MIP model. Typically, this approach requires computational time below one hour per instance. This problem reformulation is so far the best approach to solve the HHC instances considered in this thesis.

The heuristic decomposition and model reformulation proposed in this thesis can find solutions to the real-world home healthcare problem. The main achievement is the reduction of computational memory and computational time which are required by the optimisation solver. Our studies show the best way to control the use of solver memory is the heuristic decomposition approach, particularly the RDCR method. The RDCR method can find a solution for every instance used throughout this thesis and keep the memory usage within personal computer memory ranges. Also, the computational time required to solve an instance being less than 8 minutes, for which the solution gap to the optimal solution is on average 12%. In contrast, the strong point of the model reformulation approach over the heuristic decomposition is that the model reformulation provides higher quality solutions. The relative gaps of solutions between the solution for solving the reformulated model and the solution from solving the full model is less than 1% whilst its the computational time could be up to one hour and its computational memory could require up to 100 GB. Therefore, the heuristic decomposition approach is a method for finding a solution using restricted resources while the model reformulation is an approach for when a high solution quality is required. Hence, two mathematical programming based heuristic approaches are each more suitable in different circumstances in which both find high quality solutions within an acceptable time limit.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to give a special thank my supervisor Dr. Dario Landa-Silva for the continuous support, guidance and encouragement throughout my doctoral study. He always provides positive criticism in research and thesis writing. His support went beyond his academic duties. I would like to thank my second supervisor Dr. Rong Qu for her suggestions and comments which improved my work. My gratitude also goes to my viva voce examiner Dr. Chris Potts and Dr. Jason Atkin for their comments in the final version of this thesis.

This PhD research would not have been possible without financial support provided by the Institute for the Promotion of Teaching Science and Technology (IPST), the Royal Thai Government through the Development and Promotion of Science and Technology Talents Project.

My thanks go to many ASAP researchers at Nottingham. Among them: Rodrigo, Arturo, Haneen, Peng, Wenwen, Binhui, Jonata, Imo, Raras and Osman. I also thank the support provided by the administrative staff at the school of computer science, in particular Deborah Pitchfork and Christine Fletcher.

Finally, I would like to thank my parents Mr. Pharuehat Laesanklang and Mrs Wareebhorn Laesanklang for their constant support and motivation. Equally important, I would like to thank my sister Ms. Tidarat Laesanklang and my friends in Thailand.

Chapter 1

Introduction

This thesis focuses on ways to exploit mixed integer programming to solve a workforce scheduling and routing problem (WSRP). The problem is to find schedules for mobile workers to visit multiple locations. A home healthcare problem (HHC) is an example of WSRP. The home healthcare problem is to produce plans for nurses or care workers to carry out services at a patient's home [91]. The solution methods presented in this thesis are mainly developed to tackle the HHC.

1.1 Background and Motivation

The workforce Scheduling and Routing Problem (WSRP) has become especially important in recent years because the number of businesses using a mobile workforce is growing [72]. These businesses usually provide services to people at their home. Examples are home care, home healthcare, security patrol services, broadband installation services, etc. In this type of scenario, a mobile workforce must travel from its base to visit multiple locations to deliver services. The problem focuses on delivery of services, i.e. workers who perform a task must have essential skills to make the visit. The WSRP is considered as a highly constrained problem, in which a minimal change to the feasible solution is likely to generate an infeasible one [42]. As such, replacing the qualified worker with a random worker to make a visit may not be possible because the random selected worker may not have essential skills to deliver the service. In addition, most of WSRPs in the real-world are large scale problems because the nature of business is to provide services to a large number of customers which then also requires a large number of workers to deliver those services.

The literature shows that WSRP is a difficult problem [33]. Various metaheuristic methods have been applied to solve WSRP such as tabu search [58, 120], constructive heuristics [36], genetic algorithm [6, 30, 74], particle swarm optimisation [4, 5], simulated annealing [77], and variable neighbourhood search [43, 91, 103]. Using these methods has been reported to provide robust and good feasible solutions. They have reasonably low computational resource requirements, i.e. physical memory and computational time. However, implementing a heuristic method for a highly constrained problem might be difficult unless the constraints can be implemented directly to the algorithms [29, 59].

There have been attempts to use mathematical programming method to find an optimal solution for WSRP [33, 36, 107]. The problems are usually formulated as mixed integer programs (MIP) and implemented as a flow problem [22, 25, 55]. Although, the linear programming model [7] and the integer programming model [76] have also been presented. Implementing constraints into linear formulations from scratch is not easy but most of the important constraints arising in WSRP have already been published in the literature. There are two main types of real-world requirements: hard conditions and soft conditions [28]. A hard condition is a strict case in which a solution violates the condition will become infeasible or invalid. Implementing hard conditions to the MIP is strait forward, i.e. adding a linear formulation to the model as a problem constraint. A constraint creates a linear boundary in which solutions outside the boundary line are invalid, or infeasible. On the other hand, a solution violating soft condition, remains feasible but the solution is less preferred than the solution with no soft violation or having fewer soft condition violations. A soft condition can be implemented by having a surplus variable added to a linear constraint. In addition, a soft condition violation cost is added to the objective value for every unit of the surplus variable that is used. This will guarantee the lowest soft condition violation in the optimal solution. The soft condition can be seen as a goal where the condition satisfy the soft condition is desirable. A MIP model is usually tackled by MIP solvers such as IBM ILOG CPLEX, Gurobi or AMPL. However, because solving real-world problems usually requires very high computational resources, the MIP solvers are typically able to find solutions for only small instances.

Using a decomposition method is a way to extend the use of a mathematical programming method to the large scale problem. The decomposition method breaks the main problem into smaller parts which are easier to solve. For example, Dantzig-Wolfe decomposition, which works as a main decomposition for delayed column generation algorithm, splits a problem into a master problem and multiple pricing sub-problems [50]. The aim of solving a pricing sub-problem is to find a combination of visits to be made by a single worker, to which a combination is defined as a column. However, only reduced cost columns can be used in the master problem because selecting those reduced cost columns decreases the overall objective value. The master problem then decides reduced cost columns to be used in the solution in which the selected column set must provide the cheapest cost solution to the current master problem. The process iteratively solves pricing sub-problems and master problem until no further improvement can be made, i.e. no column with reduced cost found from solving the pricing sub-problems. Generally, a master problem is small and easy to solve but the pricing sub-problems become challenging and they usually have to be solved heuristically.

Heuristic decomposition is another way to use mathematical programming method to find a good feasible solution for the large problem [51, 83]. In this thesis, heuristic decomposition is an approach to find feasible solutions by breaking a problem into sub-problems, in which the decomposed problem may not cover all feasible solutions. The approach does not consider the best bound calculations to prove optimality. This approach can be considered a hybrid method as it uses mathematical programming method in the heuristic way. For example, heuristics for the generation of columns in the column generation method [25], partitioning the problem into sub-problems, and then obtaining a global solution [78], etc.

The HHC instances used in this research are scenarios from real-world problems in which a service provider delivers healthcare across the UK. The HHC can be considered a non-deterministic polynomial time hard (NP-hard) problem, because it is a combination of two NP-hard problems: the personal scheduling problem [27] and the vehicle routing problem [80]. Some of these instances are considered to be large-scale highly constrained scenarios, i.e. the largest instance involves scheduling 1,011 workers to make 1,726 visits. An assignment must consider worker availability, appointment time, visit duration, skills and qualifications, visit requirements, preferences and costs. The example is the actual operations of year 2014 where the number of customers is increasing.

In addition to the WSRP, there is a set of special constraints, called time-dependent activities constraints, representing situations in which visits have time-wise relations such as synchronising two workers to make a visit. There are several real-world examples with time-dependent activities such as groups of technicians deployed in multiple locations at the same time for cable network maintenance jobs, a doctor making a visit only when a nurse is attending the same patient, etc. This set of constraints makes the problem more difficult particularly to find a feasible solution. The time-dependent activities constraints reduce flexibility of visit assignments. These constraints are not applied to the HHC instances.

This thesis evaluates solution approaches to solve a problem instance by two main measurements: solution quality and computational time. The solution quality can be presented by the solution objective value provided by an objective function of each problem. For a minimisation problem, which applies to all problems in this thesis, a lower objective value is a better solution. The solution objective function may be compared with the optimal solution to stipulate how far the obtained solution can be improved; the relative gap to the optimal solution can be calculated by:

$$\text{Gap} = \frac{z - z^*}{|z^*|} \times 100$$

where z is the objective value of the current solution, and z^* is the value of the optimal solution. This formulation has been used widely in the MIP solver such as CPLEX. In some problem instances, the optimal solution might not be found. Thus, a modification of gap measurements can be bounded by comparing the current solution to the best known solution as given by:

$$\text{Gap} = \frac{z - z^b}{|z^b|} \times 100$$

where z is the objective value of the current solution, and z^b is the value of the best known solution. The relative gap can be normally ranged from $[0, \inf)$. This formulation has been used by Castillo-Salazar et al. [36].

This research focuses on implementing a mathematical programming model, investigating the use of a mathematical programming solver, developing heuristic decomposition approaches which harness the use of an MIP solver and finding ways to obtain a good feasible solution to a case study. In addition, this thesis also extends a heuristic decomposition method to solve the WSRP with time-dependent activities constraints.

1.2 Summary of Contributions

This thesis investigates ways to harness the use of mathematical programs on real-world problems including:

- A review of mathematical formulations used by five mathematical models. The review, presented in Chapter 2, compares formulations which could apply to WSRP. Formulations are selected to be implemented for HHC scenarios, which is used throughout this thesis as a full MIP model. Solutions given by solving the full model using the mathematical solver provides 18 benchmark results out of 42 instances. The rest were found to be too difficult to be solved optimally by the state-of-the-art mathematical solver.

- A decomposition method with conflict avoidance scheme. This proposed method decomposes a problem into sub-problems heuristically. Sub-problems are then solved in hierarchical order in order to avoid conflicting assignments. The study reveals that decomposing a problem into parts significantly reduces computational resources required by the mathematical solver. However, there are parts of the problem to be shared between sub-problems in which applying a different sub-problem solving order affects the quality of the solution. We argue that finding a sub-problem solving order to deliver the best solution would take too much permutation to find the best solving sequence and it may not exist. However, this approach is the first attempt that finds a feasible solution within 8 hours of computational time. The work has been presented in a conference paper (see Section 1.4, Publication 2) and an extended paper is to present in a book chapter (see Section 1.4, Publication 3).

- A decomposition method with conflict repair. This is an improvement to the heuristic decomposition with conflict avoidance which no longer requires to define a sub-problem solving order. Conflicting assignments are allowed when solving sub-problems. These conflicting assignments are then repaired by conflicting assignments repair. This work presents two varieties: heuristic assists (Decomposition, Conflict Repair and Heuristic assignment) or iterative procedure (iteratively used Decomposition and Conflict Repair). The decomposition with conflict repair put its computational resources mainly into the conflicting assignments which results in increasing the solution quality. However, the computational study shows that the main computational time is taken by the first iteration of solving decomposed sub-problems, which can be reduced by decreasing the sub-problem size. The study also shows the decreased sub-problem size with iterative procedure is the fastest of these decomposition approach and provides solutions with the highest quality amongst the decomposition approaches.

- A modification of the decomposition with conflict repair method to tackle WSRP with time-dependent activities constraints. The modification is an extension to support time-dependent activities constraints. A sub-problem of the decomposition step has an additional rule, in which time-dependent activities must be allocated in the same sub-problem. This results in the sub-problem solutions satisfy time-dependent activities constraints. The assigned time of these visits are forced to remain unchanged in the later process to guarantee the constraint satisfaction of the final solution. Overall, the solutions of this approach are slightly better than the greedy heuristic algorithm which is tailor-made to solve this problem.

- A proposed problem-specific mathematical model for HHC scenarios. This is a reformulation of the mathematical model based on knowledge of specific HHC scenarios. The reformulation reduces the problem complexity by simplifying constraints and restriction of the problem into a few terms. The reformulated model may discard some conditions which cause additional decision making. This reformulated model is small enough to solve by a mathematical solver. However, it requires a different data representation which compresses almost every detail from the original data. The reformulation approach provides the best solution of the HHC instances.

1.3 Structure of Thesis

Chapter 2 presents constraints and requirements of the WSRP. It shows mathematical models formulated in the literature to tackle the WSRP. The mathematical formulations are used in the implementation of the HHC instances used throughout this thesis. This chapter also describes information about HHC instances and the computational results from applying the MIP solver to those real-world scenarios. Finally, this chapter describes the amount of resources required to solve this problem by the MIP solver. The computational study shows some HHC instances can be solved optimally by the mathematical programming solver. These results are then set as benchmark results to which the quality of other heuristic solutions can be compared. However, for the other 24 instances, the mathematical programming solver requires computational memory more than 100 GB limits. An estimation shows the largest instance may require up to 24 TB of RAM. The result of the study, particularly with the 24 larger instances, reveals challenges to tackle these problem using the mathematical programming solver. This leads to investigations taken in the other following chapters.

Chapter 3 describes decomposition methods used in the literature. This chapter also presents an implementation of a decomposition method, the Dantzig-Wolfe decomposition. A computational study using the decomposition on the HHC instances is presented. This clearly shows that HHC instances are difficult as the Dantzig-Wolfe decomposition cannot bring enough of an improvement to the solution. Alternatively, the chapter also reviews the use of heuristic decomposition methods. These heuristic decomposition methods can solve the other combinatorial problems, i.e. vehicle routing problem, which motivates the development of the heuristic decomposition method to tackle HHC.

Chapter 4 proposes a heuristic decomposition, geographical decomposition with conflict avoidance approach (GDCA). This method decomposes a problem heuristically. It splits a problem into sub-problems by geographical regions. To avoid conflict, this approach requires a sub-problem solving order. Such a solving order could give a distinct solution. Therefore, we executed two experiments: applying all possible solving orders to small instances, and applying a defined solving order rules to every instance. We also present an experiment to find a sub-problem order which can produce an acceptable solution.

An extension of GDCA is also given in this chapter. The extension finds a neighbour workforce, which lives nearby the geographical region but is not available on the region, to act as reinforcements. Assignments made to neighbour workforce will result in reducing the number of unassigned visits and adding soft constraint violations. Note that the cost of an unassigned visit is less than the soft constraint violation cost. The computational results show a reduction on computational requirements in which the GDCA finds a feasible solution for every HHC instance in a time limit. The solution quality is in an acceptable range, 30% relative gap to the optimal solution on average. However, there is room for improvements in both computational time and solution quality.

Chapter 5 proposes an improved heuristic decomposition which does not require a solving order. We propose two variants of this heuristic decomposition approach: geographical decomposition with conflict repair (GDCR) and repeated decomposition and conflict repair (RDCR). This heuristic approach does not avoid conflicting assignments. Therefore, a conflicting assignments repair is introduced here to fix the conflicting assignments. However, the repair process might cancel some assignments made by decomposition. Those unassigned are tackled in two different variants: heuristic assignment in GDCA or using iterative process in RDCR. This chapter also tests an effect of applying different decomposition rules. Finally, it proposes a decomposition rule that is relatively faster and better than the conflict avoidance method.

Chapter 6 applies one of the method proposed in Chapter 5, the repeated decomposition and conflict repair (RDCR), to the WSRP with time-dependent activities constraints. We cannot apply the decomposition method directly because the conflicting assignment repair step may rearrange assignment times which potentially violate time-dependent activities constraints. Therefore, a modification to time-dependent activities constraints is applied. The first modification is made to the process of generating decomposition sub-problem, in which time-dependent activities are grouped in the same sub-problem. The solution to a sub-problem satisfies the time-dependent activities constraints. The time-dependent assignments are then fixed to preserve the time-dependent activities constraint satisfaction even if these assignments has time conflicts with the other assignments, which is then resolved by the conflicting assignments repair. A computational study compares the RDCR solutions with a greedy heuristic algorithm proposed in [36]. The results show the RDCR provides a slightly higher number of better solutions.

Chapter 7 presents another angle of problem decomposition which transforms a full model proposed in Chapter 2 into a smaller one. Thus, a compact model to solve HHC instances is proposed, in which the number of constraints is much smaller than the full model. This compact model requires a different data format in response to the different model constraints. Therefore, we also present a reformulation process to the data instance. The full data is compressed into three matrices which present 1) time conflicting between visits, 2) compatibility of workforce to take visits, and 3) the total assignment costs. The compact model is then solved to optimality by a mathematical programming solver. The solution to the small model is then transformed back to the solution format supported by the full model so that comparison between algorithms can be made.

Chapter 8 sums up the contribution of this thesis and presents research directions on this topic for future research.

1.4 List of Publications

1. Wasakorn Laesanklang and Dario Landa-Silva. **Mixed-integer programming for the workforce scheduling and routing problem.** In *International Conference on Applied Mathematical Optimization and Modelling (APMOD 2014)*, pp. 34, 2014, Abstract Only.

2. Wasakorn Laesanklang, Dario Landa-Silva and J. Arturo Castillo-Salazar. **Mixed Integer Programming with Decomposition to Solve a Workforce Scheduling and Routing Problem.** In *Proceedings of the 4th International Conference on Operations Research and Enterprise Systems (ICORES 2015)*, pp. 283–293, Scitepress, Lisbon, Portugal, January 2015, Best Student Paper Award.

3. Wasakorn Laesanklang, Rodrigo Lankaites Pinheiro, Haneen Algethami

and Dario Landa-Silva. **Extended Decomposition for Mixed Integer Programming to Solve a Workforce Scheduling and Routing Problem.** In *Operations Research and Enterprise Systems*, Series Communications in Computer and Information Science, Vol. 577, pp. 191–211, Springer, 2015.

4. Wasakorn Laesanklang, Dario Landa-Silva and J. Arturo Castillo-Salazar. **Mixed Integer Programming with Decomposition for Workforce Scheduling and Routing With Time-dependent Activities Constraints.** In *Proceedings of the 5th International Conference on Operations Research and Enterprise Systems (ICORES 2016)*, pp. 283–293, Scitepress, Rome, Italy, February 2016.

5. Wasakorn Laesanklang and Dario Landa-Silva. **Decomposition Techniques with Mixed Integer Programming and Heuristics to Solve Home Healthcare Planning Problems.** *Annals of Operations Research*, doi:10.1007/s10479-016-2352-8, 2016.

6. Wasakorn Laesanklang, Dario Landa-Silva and J. Arturo Castillo-Salazar. **An Investigation of Heuristic Decomposition to Tackle Workforce Scheduling and Routing With Time-dependent Activities Constraints.** submitted, to be appear in *Operations Research and Enterprise Systems*, Series Communications in Computer and Information Science.

Chapter 2

Mixed Integer Programming for a Workforce Scheduling and Routing Problem

This chapter focuses on the problem to be solved in this thesis. The main problem is a home healthcare problem (HHC) which is a variant of a workforce scheduling and routing problem (WSRP). The HHC problem has almost the same component as the WSRP, except that it has fixed time windows and does not have time-dependent activities constraints. HHC instances are described in Section 2.5. Methods proposed in this thesis all aim to to solve the HHC problem, except methods in Chapter 6 which aim to solve the WSRP with time-dependent activities constraints (more details inside the chapter).

This chapter provides background knowledge for the WSRP, a review of the literature for WSRP mathematical models, the HHC problem and our implemented MIP model, the HHC instances to be used throughout this thesis, and its solutions producing by a MIP solver.

The MIP model in this chapter has been presented in the following papers:

- Wasakorn Laesanklang, Rodrigo Lankaites Pinheiro, Haneen Algethami and Dario Landa-Silva. **Extended Decomposition for Mixed Integer Programming to Solve a Workforce Scheduling and Routing Problem.** In *Operations Research and Enterprise Systems*, Series Communications in Computer and Information Science, Vol. 577, pp. 191–211, Springer, 2015.

- Wasakorn Laesanklang and Dario Landa-Silva. **Decomposition Techniques with Mixed Integer Programming and Heuristics to Solve Home Healthcare Planning Problems.** *Annals of Operations Research*, online-first, 2016.

2.1 Workforce Scheduling and Routing Problem

The Workforce Scheduling and Routing Problem (WSRP) is to address the scheduling of mobile personnel visiting different locations [35]. Examples of WSRP Scenarios include home healthcare, home care, scheduling technicians, security personnel routing and rostering, and manpower allocation. An assumption when defining a problem to be WSRP is that the workforce should spend more time doing work than travelling. Therefore, the focus of the business is to deliver the right services to its customers.

Table 2.1 presents WSRP characteristics and their definition which are found in the literature. The first column shows types of characteristic and the second column presents a definition of each characteristic. There are 7 characteristics which are summarised by Castillo-Salazar et al. [35]: time windows, skills and qualifications, service time, start and end locations, connected activities, and teaming.

1. Time Windows

   A time window indicates the time by which the activity must start [118]. The values are commonly presented as the earliest starting time and the latest starting time for each visit. A visit to be made must start within the

Table 2.1: WSRP constraints in the literature

Constraint	Definition
Time Windows	A time interval for starting a visit. Workforce can start the work as soon as they reach the working location in between the interval. Time windows can be flexible or tight depending on problem requirements. An exact time window is also possible, i.e. a visit must start at the appointment time.
Skills and Qualifications	Only qualified workforce can work on a visit which requires primary skills. Generally, an organisation has diversity of skilled workforce. Hence, assigning under-skilled workforce is prohibited. Some cases also require the minimisation of assigning overqualified workforce as they should be preserved for the high skill requirement only.
Service Time	A duration of a working visit. In reality, the duration is very dependent on an individual worker. In practice, the service time is assumed to be of a fixed duration.
Start and End Locations	Workforce may (leave from/return to) a single starting location (office), or many locations (i.e. from their home). Starting location and ending location may be defined as different places.
Connected Activities	Two or more visits may depend on each other. It includes sequential dependency (a visit must be performed before the other), synchronisation (visits start at the same time), overlap (the second visit starts while another visit is in progress) and dependency with time differences (sequenced visits with a break interval and/or expired time before starting the next visit).
Teaming	Visits require a group or team to participate. Problem of having fixed teams, because they are not changed for the whole plan. This case may define a team as a single worker. The other cases show team may be changed during the time horizon. For example, a worker may join a team during his morning visit and join the other team in the afternoon.
Clusterisation	Visits are grouped in clusters or zones. It may apply to prevent assigning a worker long distances to travel. It also can be used to reduce the size of the problem by tackling sub-problems instead of the whole problem.

time interval. The time window interval shows a flexibility of the visit. A visit with time-wised priority usually has a narrower time window interval, e.g. a visit for patient to have medicine should have 10 minutes of time window interval. In some cases, the time window interval has length 0, i.e. the earliest start time is equal to the latest start time, which is called an *exact time window* [61]. This exact time window case appears in the home healthcare scenarios which are used throughout this thesis.

2. Skills and Qualifications

   Skills and qualifications are values to narrow the candidate workers down. In this case, a visit must be made by workers who have the required skills. A problem might define workers with no differences in skills, called uni-skill worker [15, 67, 75, 127]. The uni-skill problem usually is a simplified case which only arranges the number of workers for each working shift. However, problems related to the real-world usually have multiple skills. The hierarchical skill is when higher ranked skills can substitute lower ranked skills, and the reverse is not valid [19, 114, 122]. The worker with higher ranked skills is known as a specialist and the worker with lower ranked skills is defined as a generalist. Assigning an over qualified worker might result in penalty costs in the proposed solution. The multi-skill case is when two different skills cannot be replaced by each other. A job requirement may state a combination of skills [64, 70, 81]. However, the real world problem usually has a combination of multiple and hierarchical skills [38]. In summary, workforce scheduling is to allocate qualified workers for jobs that have certain skills demands.

3. Service Time

   A service time is a duration that workers must spend when they make a visit. Generally, the service time is tackled as a fixed value for each visit.

However, the duration might be defined based on worker's skills, e.g. a worker with higher ranked skills should take less time on a visit compared to a worker who has lower ranked skills. This latter case is rarely found in the literature because it adds difficulties of the problem and some cases may require workers to attend a visit for the whole duration.

4. Start and End Locations

   Workers can start and end their journey at any location depending on the type of the problem. A start and end location can be a single point, called a single depot problem [37, 40, 79, 118]. The problem can be extended to a multiple depots problem when a worker starts and finishes their journey at the same location but different workers may have different depots [48, 53, 88]. An example is the case that workers leave their home for work and finally finish the day by returning to their home. The other case is the combination of single depot and multiple depot, i.e. workers must start their work from the central depot but they can go straight to their home after they complete the last visit.

5. Connected Activities

   This characteristic explains a visit that may depend on another visit. Connected activities may be defined in a time-based restriction, known as *time-dependent activities* [107]. There are five types of time-dependent activities: synchronise, overlap, minimum difference, maximum difference, and min-max difference. The time-dependent activities will be further explained in *time-dependent activities constraints* which appears later in this chapter.

6. Teaming

   Some visits may require a team due to the nature of the work [82]. Team members may remain unchanged throughout the planning horizon which a whole team is can be considered as a single person. Although, the generalised problem should consider temporally teams, i.e. a team is formed just for a required visit and its members can travel to different locations and continue on to other visits. This case might be considered as a group of synchronised visits which requires multiple visits to be made at the same time.

7. Clusterisation

   Visits might be grouped when they are located in the same regions e.g. the same building, the same street or the same county. A reason behind clusterisation is that workers usually not prefer to work too far away from their home. As a result, workers may choose a set of regions they prefer. Additionally, clusterisation might be used to reduce the number of visits by considering a group as a single visit location which then decreases the problem difficulty.

WSRP scenarios may have their specific features depending on their real-world application. We choose home healthcare (HHC) scenarios as an example, with its requirements given by our industrial partner. We remind the reader that the HHC problem is to allocate care workers to make visits at to homes of the patients. In practice, patients or customers usually order regular visits, e.g. a visit every Monday at 10 AM. We note that this problem is an exact time window problem.

Each visit requests workers with multiple skills which can be expressed into two sets: minimum skill requirements and additional skill requirements. Workers who will make the visit must at least have the minimum skill requirements and workers who have the additional skills are preferable. A patient may request a team to make visits. For this problem, temporally team approach is applied, i.e. a nurse and a doctor are met at patient home, so the team can be split thereafter. A visit requires a fixed service time, i.e. workers must stay with the patient for the whole duration.

The HHC is a multiple depots problem because care workers prefer to leave for work from their home. The problem also assumes that workers return to their home after they finish their tour. The problem also clusters visits into geographical regions. Workers also have their responsibility regions and preferred regions. The scheduler should assign visits located in worker's responsibility regions. However, the problem does not take this as a strict condition because realistically a worker can make visits outside their responsibility regions. Workers also have their working times so visits assigned to workers should lie within their working period. However, workers might be requested to make visits outside their working times. We note that some visits may be left unassigned due to lack of skilled workers. The value of unassigned visits are very important to our industrial partner to estimate their limitation and which possibly a future improvement to their services.

A solution to the WSRP is evaluated by multiple criteria, for example: travelling distances, travelling times, operational costs, workforce efficiency, workforce/client preferences and the number of unassigned visits. These multiple aspects can be tackled as a multi-objective problem [9, 18]. The multi-objective approach finds multiple solutions and leave decision makers to choose which solution they will use [24]. These solutions must not be completely dominated by the other solutions, i.e. all quality measure values of the completely dominated solution are lower than a dominating solution. This approach requires a large computational time to provide a set of non-dominated solutions. Alternatively, a single objective approach can be used if the decision maker has a rule for decision making. The rule is then converted to a mathematical function, called weighted-sum, which is a summation of the weighted quality measure value, where the weights are provided by decision maker.

Table 2.2: Relation between WSRP conditions/requirements and WSRP constraints to be implemented in MIP model

Conditions/Requirements	To be appear in
Network Graph & Teaming Characteristic	Visit Assignment Constraint
Network Graph	Route Continuity Constraint
Network Graph & Service Time	Travel Time Feasibility Constraint
Time Window Characteristic	Time Window Constraint
Time Window Characteristic	Workforce Time Availability Constraint
Skills and Qualification Characteristic	Skills and Qualifications Constraint
Start and End Location Characteristic	Start and End Locations Constraint
Connected Activities Characteristic	Special Case: Time-dependent Constraints
Clusterisation Characteristic	Working Region Constraint

2.2 Literature Review

The problem characteristics from the previous section leads us to the WSRP constraints and their implementation. Before explaining the details of each constraint, this section starts with explanation of general concept of mathematical models.

MIP models defining the WSRP are usually formulated as a flow model [26]. Generally, a directed graph $G = (V, E)$ represents the network flows, where V is a set of nodes to represent visits and start-end locations and E is a set of edges between nodes which each edge presents a travel route between two visits. The problem is to find paths along edges that set off from starting nodes, then pass the visiting nodes and reach the ending nodes, while maximising the number of nodes to be visited. The maximum number of paths is equal to the number of workers. The network model has been applied to other problem such as scheduling problems, and routing problems. In addition, constraints of the scheduling problem and the routing problem can be adopted for the WSRP because they share the graph structure.

Problem characteristics and the network structure are defined as constraints in MIP models as shown in Table 2.2. Table 2.2 presents two columns to relate the problem characteristics and the network structure of the problem with the constraints implemented in the MIP model. In mathematical programming problem, constraints are treated as boundaries of search space in which feasible solutions are located inside the border (including the border line), known as feasible regions. These constraints are generally explained in mathematical formulations. The objective function can be used to evaluate the solution quality. Both objective function and constraints are very important in a mathematical programming problem [23].

There are common constraints where most of the MIP models in the literature are implemented and problem specific constraints which are defined to specific problem requirements. We argue that an integration of constraints presented in the literature might cover the most of existing real-world requirements. In this thesis, we describe details of five selected mathematical models from the literature: Bredstrom and Ronnqvist [26], Rasmussen et al. [107], Dohn et al. [55], Trautsamwieser and Hirsch [121], and Barrera et al. [13].

Table 2.3 lists the notation of sets, parameters, and variables for explaining mathematical models in the literature, and the proposed mixed integer programming model presented later in this chapter. This notation will be used throughout this thesis. The domain of notation presented in this table is presented based on the proposed mixed integer programming model to solve HHC problem which is explained in Section 2.4. Here, we use the same notation in every model to present the notations with the same meaning. Although, there might be differences in their domains, i.e. y_j is a binary variable in [107], but y_j in our implemented model is an integer variable. This is because models in the literature have different implementation concepts.

Table 2.3: Notation used in MIP model for WSRP

Sets	Definition
V	Set of all nodes denoted by $V = D \cup T \cup D'$. Indices $i, j \in V$ instantiate nodes.
D	Set of source nodes, i.e. starting locations.
D'	Set of sink nodes, i.e. ending locations.
T	Set of visiting nodes.
V^S	Set of nodes have leaving edges, i.e. $V^S = D \cup T$.
V^N	Set of nodes have entering edges, i.e. $V^N = D' \cup T$.
E	Set of edges connecting pairs of nodes.
K	Set of workers, k is a worker in K.
S	Set of dependency visits. Members are pairs of visit (i, j) in which visit i and j are dependent.
Parameters	
M	Large constant.
$\lambda_1, \ldots, \lambda_4$	Objective weights.
$t_{i,j} \in \mathbb{R}^+$	Travelling duration between node $i \in V^S$ and node $j \in V^N$.
$d_{i,j} \in \mathbb{R}^+$	Travelling distance between node $i \in V^S$ and node $j \in V^N$.
$p_j^k \in \mathbb{R}^+$	Cost of assigning worker k to node $j \in T$.
$\rho_j^k \in \mathbb{R}^+$	Preferences value of assigning worker k to node $j \in T$.
$r_j \in \mathbb{N}$	The number of required workers at node $j \in T$.
$\delta_j \in \mathbb{R}^+$	Duration of visit at node $j \in T$.
$\alpha_L^k, \alpha_U^k \in \mathbb{R}^+$	Shift starting and ending time for worker k.
$w_j^L, w_j^U \in \mathbb{R}^+$	Lower and upper time windows to arrive node j.
$v_j^L, v_j^U \in \mathbb{R}^+$	Lower and upper soft time windows to arrive node j.
$h^k \in \mathbb{R}$	Maximum working duration for worker k.
$\eta_j^k \in \{0, 1\}$	Qualification of worker k at node j, the value is 1 when a worker is qualified to work, 0 otherwise.
$\gamma_j^k \in \{0, 1\}$	Worker region availability on node j, the value is 1 when a worker is available in the region of visit j, 0 otherwise.
$s_{i,j} \in \mathbb{R}$	Dependency coefficient. The value states relation of visit i and visit j when $(i, j) \in S$.
$Q^k \in \mathbb{R}$	Skill proficiency levels of worker $k \in K$
$q^j \in \mathbb{R}$	Minimum qualification levels required to make a visit $j \in T$
Variables	
$x_{i,j}^k \in \{0, 1\}$	Worker assignment decision variable, the value is 1 when a link between $i \in V^S$ and $j \in V^N$ is assigned to worker k, 0 otherwise.
$\omega_j \in \{0, 1\}$	Working shift violation indicator variable, the value is 1 when the assignment at node j is made outside working shift, 0 otherwise.
$\psi_j \in \{0, 1\}$	Worker's region violation indicator variable, the value is 1 when the assignment at node j is violated, 0 otherwise.
$y_j \in \mathbb{N}$	Unassigned visit indicator variable, the value is 1 when assignment does not include at node j.
$a_j^k, \tilde{a}_j^k \in \mathbb{R}^+$	Arrival time decision variable for worker k to start work at node j. Note that a_j^k can be any number when worker k is not assigned to node j, but $\tilde{a}_j^k = 0$.

2.3 Constraints for Workforce Scheduling and Routing Problem in the Literature

This section analyses constraints implemented on the five selected mathematical models listed above. For simplicity, we use the following short-hand notation to refer to each of the five works:

- ODS-HHC: Optimisation of Daily Scheduling for Home Health Care Services [121],
- NB-TCS: A Network-based Approach to the Multi-activity Combined Timetabling and Crew Scheduling Problem: Workforce Scheduling for Public Health Policy Implementation [13],
- VRS-TPS: Combined Vehicle Routing and Scheduling with Temporal Precedence and Synchronization Constraints [26],
- MAP-TTC: The Manpower Allocation Problem with Time windows and Job-teaming Constraints [55], and
- HCS-PCD: The Home Care Crew Scheduling Problem: Preference-based visit clustering and temporal dependencies [107].

Generally, each paper defines its own set of notations to explain its mathematical model. However, to make comparisons between models, the notations presented in this thesis are normalised to the same set presented in Table 2.3.

Each constraint is presented individually to compare the five implementation approaches.

2.3.1 Visit Assignment Constraints

This constraint indicates that visits require a worker. It is the backbone of many problems as it pairs workers to attending visits. Table 2.4 compares the visit

Table 2.4: Visit assignment constraint comparison between five different mathematical models.

	Visit assignment constraint	
ODS-HHC	All visits need a worker to visit. No unassigned visit allowed.	
	Hard constraint:	$\sum_{i \in V^S} \sum_{k \in K} x_{i,j}^k = 1 \; \forall j \in T$
NB-TCS	All demands need to be filled. Only qualified workforce indicated by a binary parameter can be selected. Assigned visit must be balanced amongst workforce. b is a variable for maximum workload differences between workers.	
	Hard constraint:	$\sum_{k \in K} \sum_{i \in V^S} \eta_j^k x_{i,j}^k = r_j \; \forall j \in T,\; \eta_j^k,\; r_j$ are binary
	Balance assignment:	$\sum_{i,j \in V} \delta_j x_{i,j}^{k_1} - \sum_{i,j \in V} \delta_j x_{i,j}^{k_2} \leq b \quad \forall k_1, k_2 \in K : k_1 \neq k_2$
	Balance objective:	Minimise b
VRS-TPS	All visits need a visit. No unassigned visit allowed. b is a variable for maximum workload differences between workers.	
	Hard constraint:	$\sum_{k \in K} \sum_{i \in V^S} x_{i,j}^k = 1 \; \forall j \in T$
	Balance assignment:	$\sum_{i,j \in V} \delta_j x_{i,j}^{k_1} - \sum_{i,j \in V} \delta_j x_{i,j}^{k_2} \leq b \; \forall k_1, k_2 \in K : k_1 \neq k_2$
	Balance objective:	Minimise b
MAP-TTC	Visiting must not exceed demand. Note that objective is to maximise the number of assignments made.	
	Hard constraint:	$\sum_{k \in K} \sum_{i \in V^S} x_{i,j}^k \leq r_j \; \forall j \in T$
	Obj. function:	$\max \sum_{k \in K} \sum_{i \in V^S} \sum_{j \in V^N} x_{i,j}^k$
HCS-PCD	Soft constraint where unassigned visits are charged in objective function.	
	Soft constraint:	$\sum_{i \in V^S} \sum_{k \in K} x_{i,j}^k + y_j = 1 \; \forall j \in T$
	Obj. function:	$\text{Min} \sum_{j \in T} y_j$

assignment constraint for the five mathematical models.

This constraint can be implemented by simply stating that every visit needs exactly one worker as in models ODS-HHC and VRS-TPS. These two models consider visit assignment as a hard condition where all visits must be made. On the other hand, a soft condition implementation of this constraint can be done as presented in model HCS-PCD. As such, visits are allowed to be unassigned but need to be minimised. Models NB-TCS and MAP-TTC tackle this constraint by stating visiting demand explicitly such as that visit j must have b_j workers to visit.

The assignment may also require balancing the workload amongst workers as in models NB-TCS and VRS-TPS. These models introduced additional decision variable b which is the maximum number of assignments per worker. The value needs to be minimised which ideally gives a solution with a balanced workload.

The visit assignment constraint has been implemented in the same direction, i.e. entering edges of a visiting node must be selected. The number of entering edges to be selected is equal to the number of visiting demand. The hard condition interpretation of the visiting constraint is suitable for problems which have been shown that all visits can be logically made, i.e. the number of skilled workers is sufficient to all visits. The interpretation which suitable for the real-world problems considered here is the soft condition interpretation where unassigned visits are allowed. A solution with unassigned visits could reflect causes of problems in operations such as overbooking, worker shortages, or skilled worker shortages. Therefore, the constraint to be implemented for a general WSRP is required to support the multiple visiting demand problem which is implemented as a soft condition (see Section 2.4.1). This results in mixing constraints of two models NB-TCS and HCS-PCD.

2.3.2 Route Continuity Constraints

This is commonly defined as flow conservation constraint and it states that the number of entering flows must be equal to the number of leaving flows. In the WSRP context, the number of flows refers to the number of visiting work-

Table 2.5: Route continuity constraint implemented on five mathematical models.

	Route continuity constraint.
ODS-HHC NB-TCS VRS-TPS MAP-TTC HCS-PCD	Flow conservation constraint. At a visit, the number of entering workers must be equal to the number of leaving workers. Hard constraint: $\sum_{i \in V^S} x^k_{i,h} = \sum_{j \in V^N} x^k_{h,j} \; \forall h \in T, \forall k \in K$

ers. Hence, the route continuity constraint states that the number of workers arriving to a visit must be the same as the number of workers leaving the visit. Table 2.5 shows the mathematical formulation used in all the five mathematical models to implement this constraint. The same constraint is used in our WSRP model (see Section 2.4.2).

The route continuity constraint presented by all the five models is a typical flow conservation constraint. The mathematical formulation shown in the table is in the compact form to describe this constraint which has been shown to be efficient as we can find it implemented in the five models. However, this constraint alone may not enforce a working path, for example, a cycle $x^k_{i,i} = 1$ is feasible by this constraint where the cycle is not satisfied the WSRP because the cycle does not make progress from starting location and terminate at the ending location. Therefore, cycle cases are eliminated from the WSRP solution by travel time feasibility constraints (see 2.3.4). In addition, a complete path requires a starting location and an ending location where those locations are defined in constraint start and end location (see 2.3.3). However, the route continuity constraint is the only formulation to define links between visits which is a backbone of the solution.

2.3.3 Start and End Locations Constraints

Start and end locations are general requirements for flow models. They are special nodes which only have one direction to connect to other nodes. Strictly speaking, the start location has only leaving edges and the end location has only entering edges. They are places for distributing workers and collecting them when they finish their journey. Table 2.6 shows the mathematical formulation of start and end locations on the five selected models. There are implementations in both single central location and multiple locations cases. The single central location has only one centre point to distribute all workers. The multiple locations case is when workers can start their journey from their chosen place, i.e. their home.

Most models implement this constraint by forcing all workers to leave from starting locations and return back to the ending location. Only ODS-HHC does not have this approach, so that using all workers is not required. These conditions were subject to requirements of each model.

A single central location problem, as shown in models VRS-TPS, MAP-TTC and ODS-HHC, assumes there is only one location for the start and end of a worker's route. It applies to general cases where workers need to visit their office before being deployed for work. We denote 0 is an index to represent the central location in models ODS-HHC and MAP-TCC. The constraint means a worker must leave the start location only/at most once. The same requirement applies to end location. Note that the model VRS-TPS shows the formulation for multiple depots but the problem instances tackled in this work only considered a single location.

The NB-TCS model also defines a single location problem. However, the assignment constraints include the condition to control assignments from start to finish. As such, there is no explicit implementation of this condition.

The HCS-PCD model has multiple start and end locations. The case represents a problem with multiple offices or if workers are able to start their journey from their home. The implemented constraint only applies to worker k and their selected start location and edges connecting between the worker k and the

Table 2.6: Start and end locations implemented by five mathematical models.

	Start/End location	
ODS-HHC	Single depot problem.	
	Hard constraint:	$\sum_{j \in T} x^k_{0,j} \leq 1 \; \forall k \in K$ $\sum_{j \in T} x^k_{j,0} \leq 1 \forall k \in K$
NB-TCS	See NB-TCS assignment constraint.	
VRS-TPS	Single starting/ending location, all workers must be used. $\lvert D \rvert = \lvert D' \rvert = 1$	
	Hard constraint:	$\sum_{j \in T} \sum_{0 \in D} x^k_{0,j} = \sum_{j \in T} \sum_{0 \in D'} x^k_{j,0} = 1 \; \forall k \in K$
MAP-TTC	Single depot problem, all workers must be used.	
	Hard constraint:	$\sum_{j \in T} x^k_{0,j} = 1$
HCS-PCD	Multiple starting/ending locations, all workers must be used. Each worker has his own start visit and end visit. This case presents $\lvert D \rvert = \lvert D' \rvert = \lvert K \rvert$.	
	Hard constraint:	$\sum_{j \in T} x^k_{i,j} = 1, \; \forall k \in K, \exists i \in D$ $\sum_{j \in T} x^k_{j,i} = 1, \; \forall k \in K, \exists i \in D'$

other start locations are excluded by this constraint. The same condition also applies to end location.

The start and end locations constraint is an essential component in the flow model because a flow requires at least one place to start, and a place to end. The constraint defining multiple start and end locations is the approach towards the generalised constraint. The variant of constraints implemented across the five models is not too different. There is a use of inequality in model ODS-HHC which allows solutions to have unemployed workers, a worker has not been used throughout the planning horizon, i.e. a worker k has $\sum_{i,j \in T} x^k_{i,j} = 0$. In this case, the solution can produce an empty path for that worker. Although, the other models tackle the unemployed workers in a slightly different way by having a path to leave from worker start location and to connect strait to their end location without making visits. However, these two approaches finally give the same outcome.

In our implementation, we apply start and end location constraints from HCS-PCD to our HHC model with a small modification, replacing $=$ with $\leq$ to allow the case that the solution does not use some workers (see Section 2.4.3).

2.3.4 Travel Time Feasibility Constraints

This constraint guarantees time feasibility between two visits. Generally, MIP models only define a decision variable as a time stamp for each visit, mostly by defining arrival times. An arrival time a_j^k at a visit j must be feasible when considering traveling time from the predecessor location, assumed to be visit i. Here, there are models which use a slightly different principle to define arrival time, we denote the variable to be $\tilde{a}_j^k$. The difference between a_j^k and $\tilde{a}_j^k$ is that a_j^k can be any number when the visit j is not assigned to worker k, but $\tilde{a}_j^k$ must only be 0. The earliest time to leave the predecessor location is the summation of arrival time a_i^k at the predecessor location i and the working duration δ_i at that location. Therefore, the arrival times a_j^k (or $\tilde{a}_j^k$) of visit j should be at least δ_i plus travel time $t_{i,j}$ between i and j.

Table 2.7 shows the formulations implemented in the five models except only NB-TCS which does not have this constraint. The other models formulate this constraint in the same direction. They use predecessor visit i as a reference point. The arrival time a_j^k (or $\tilde{a}_j^k$) of visit j must be assigned after the summation of a_j^k or $\tilde{a}_j^k$ (arrival time of visit i), δ_i (duration to spend on visit i) and $t_{i,j}$ (travelling time between visit i and visit j). The constraint only applies to the active assignment where $x_{i,j}^k = 1$ since $M(1 - x_{i,j}^k) = 0$. Deactivated constraint, $x_{i,j}^k = 0$, leave the constraint become always valid when M is a large positive number. Model VRS-TPS uses w_i^U (the latest time availability of worker k) as a big value instead of M.

Table 2.7: Travel time feasibility constraint implemented on five mathematical models.

Visit starting time feasibility and visiting duration guarantee		
ODS-HHC HCS-PCD	Assigned arriving time guarantees a visit start after finishing its preceding visits and travelling time.	
	Hard constraint:	$\tilde{a}_j^k \geq \tilde{a}_i^k + t_{i,j} + \delta_i - M(1 - x_{i,j}^k)\ \forall i \in V^S, \forall j \in T, \forall k \in K$
MAP-TTC	Assigned arriving time guarantees a visit start after finishing its preceding visits and travelling time.	
	Hard constraint:	$a_j^k \geq a_i^k + t_{i,j} + \delta_i - M(1 - x_{i,j}^k)\ \forall i \in V^S, \forall j \in T, \forall k \in K$
NB-TCS	None defined	
VRS-TPS	Assigned arriving time guarantees a visit start after finishing its preceding visits and travelling time.	
	Hard constraint:	$\tilde{a}_j^k \geq \tilde{a}_i^k + (t_{i,j} + \delta_i)x_{i,j}^k - w_i^U(1 - x_{i,j}^k)\ \forall i \in V^S, \forall j \in T, \forall k \in K$

The travel time feasibility constraint is a formulation to connect a binary decision variable $x_{i,j}^k$ and a non-negative variable a_j^k (or $\tilde{a}_j^k$). This constraint not only provides an arrival time of worker k at location of visit j but also eliminates assignment cycles, because the assignment $x_{i,j}^k = 1$ can only be valid when arrival time a_j^k (or $\tilde{a}_j^k$) at a visit j is more than arrival time a_i^k (or $\tilde{a}_i^k$) at its predecessor visit i since $\delta_i > 0$. If there is an assignment cycle, the visit j must be assigned again after the visit i which results in $a_i^k > a_j^k$ (or $\tilde{a}_i^k > \tilde{a}_j^k$) , contradicting the previous sentence. Therefore, the travel time feasibility constraint is another important part of the models to force directions of the solution paths in one direction.

In our implementation, which is presented in Section 2.4.4, the formulation of this constraint is the same as MAP-TTC model. This is to guarantee that if a worker k is to make visit j after visit i, then the arrival time a_j^k at visit j must have enough time δ_i to fully complete jobs at visit i, and to travel $t_{i,j}$ between the two locations.

2.3.5 Time Window Constraints

This constraint limits arrival times which must lie within the given time windows. Generally, a time window means that the arrival time a_j^k of a worker k to visit j must be allocated between the earliest arrival time w_j^L and the latest arrival time w_j^U.

There are two model principles regarding the arrival time. The first principle, denoted arrival time to be a_j^k, allows the arrival time a_j^k to be any number when a visit j is not assigned to a worker k. This principle applies to model MAP-TTC. Therefore, the arrival time a_j^k to be used in a solution must be associated with $x_{i,j}^k = 1, \exists i \in V^S$. The other principle, denoting arrival time as $\tilde{a}_j^k$, forces arrival time $\tilde{a}_j^k = 0$ when a visit j is not assigned to a worker k. This case applies to models ODS-HHC, VRS-TPS, and HCS-PCD.

Table 2.8 presents implementations on the time windows constraint. NB-TCS does not implement this constraint explicitly. For the other models, all hard constraints are implemented in the same approach as:

$$w_j^L \leq a_j^k \leq w_j^U$$

However, ODS-HHC, VRS-TPS and HCS-PCD require arrival time $a_j^k = 0$ when a visit j is not assigned to a worker k. Therefore, they apply the variable $x_{i,j}^k$ to the time window parameters. Hence, the constraint becomes

$$w_j^L \sum_{i \in V^S} x_{i,j}^k \leq \tilde{a}_j^k \leq w_j^U \sum_{i \in V^S} x_{i,j}^k$$

If worker k has not been used for visit j, then $\tilde{a}_j^k = 0$ as $\sum_{i \in V^S} x_{i,j}^k = 0$.

ODS-HHC also implements soft constraints for time windows. Hence, ODS-HHC has two levels of time windows where the assignment incurs no penalty if the arrival time is allocated within the preferred time slot $[v_j^L, v_j^U]$. However, it is

Table 2.8: Time window constraint implemented on five mathematical models.

	Time window constraint	
ODS-HHC	Implemented both hard and soft constraints. A worker arrival time must lie within the visiting hard time window when the visit is allocated to a worker. Furthermore, the arrival time is preferred to be within soft time windows. The violation on soft constraint is charged to the objective function.	
	Hard constraint:	$w_j^L \sum_{i \in V^S} x_{i,j}^k \leq \tilde{a}_j^k \leq w_j^U \sum_{i \in V^S} x_{i,j}^k$
	Soft constraint:	$v_j^L - s_j^1 \leq \sum_{k \in K} \tilde{a}_j^k$ $\sum_{k \in K} \tilde{a}_j^k \geq v_j^U + s_j^2$
	Soft cons. obj.:	$\text{Min} \sum_{j \in T} s_j^1 + s_j^2, s^1, s^2 \geq 0$
NB-TCS	None defined	
VRS-TPS	Hard constraint implementation. A worker arrival time must lie between time window when the visit is allocated to a worker.	
	Hard constraint:	$w_j^L \sum_{i \in V^S} x_{i,j}^k \leq \tilde{a}_j^k \leq w_j^U \sum_{i \in V^S} x_{i,j}^k$
MAP-TTC	Hard constraint implementation. A worker arrival time must lie between time window.	
	Hard constraint:	$w_j^L \leq a_j^k \leq w_j^U$
HCS-PCD	Hard constraint implementation. A worker arrival time must lie between time window when the visit is allocated to a worker.	
	Hard constraint:	$w_j^L \sum_{i \in V^S} x_{i,j}^k \leq \tilde{a}_j^k \leq w_j^U \sum_{i \in V^S} x_{i,j}^k$

possible to allocate arrival time outside the preferred time but cannot exceed the strict time windows $[w_j^L, w_j^U]$. Duration outside the preferred time slot $s_j^1 + s_j^2$ is charged as an artificial cost to the objective function. The objective function is to minimise the time differences from the preferred time windows.

The implementations of time window constraint are presented in the similar direction. Those hard constraint implementations work in almost the same way. The most simplified formulation is a constraint in model MAP-TTC. Real world application might prefer both hard and soft condition to be implemented as presented in ODS-HHC. An additional condition can be added to the constraints; for example, an arrival time $a_j^k = 0$ when the visit j belongs to the other workers which can use mathematical formulation in models HCS-PCD and VRS-TPS. However, those arrival time will not appear in the final solution because the assignment of visit j does not belong to the worker k. Therefore, the formulation in MAP-TTC should be a welcome choice because it produces simple constraints. The time window constraint can conflict with a workforce time availability constraint. We give explanations and examples in Section 2.4.5.

2.3.6 Skills and Qualifications Constraints

This constraint defines that an assignment can be made by only qualified worker. Each visit sets a minimum qualification level for each required skill in which only qualified workers can make that visit. This constraint is usually required as a hard condition, i.e. a worker who does not pass the minimum qualification level cannot make the visit. Table 2.9 shows how this constraint is implemented in the five mathematical models.

There are two main different approach to implement this constraint as expressed above. The first approach, shown in ODS-HHC, leaves all decision to the mathematical solver by providing worker skill proficiency levels Q^k and visit minimum qualification levels q^j. A qualified worker is the one who has proficiency levels higher than the required qualification. The second approach, found in MAP-TTC and HCS-PCD, transforms both proficiency levels and minimum qualification level into a binary parameter η_j^k in which the value is 1 when a worker k is qualified to make a visit j, otherwise the value is 0.

Both implementation approaches work in the same way. However, a reason for ODS-HHC take the first constraint approach is the over-qualified level is applied to the objective function. In addition, this approach may require additional matrices to store data if the instance is a multiple skill problem, i.e. a workforce-skill matrix to define proficiency level on every skill and a visit-skill

Table 2.9: Skill and qualification constraint implemented by five mathematical models.

	Skill and qualification	
ODS-HHC	Over-qualification to be minimised in objective function. Worker skill (Q^k) must higher than visit requirement (q^j).	
	Hard constraint:	$x^k_{i,j}q_j \leq Q^k \ ,\forall i \in V^S, \forall j \in T, \forall k \in K$
	Objective function:	$\sum_{i \in V^S} \sum_{j \in T} \sum_{k \in K: q_j < Q^k} \delta_j x^k_{i,j}$
NB-TCS	See NB-TCS visit assignment constraint, Table 2.4.	
VRS-TPS	None defined	
MAP-TTC	Minimum skill guaranteed as hard constraint. Worker k is qualified when $\eta^k_j = 1$.	
	Hard constraint:	$x^k_{i,j} \leq \eta^k_j \ ,\forall i \in V^S, \forall j \in T, \forall k \in K$
HCS-PCD	Minimum skill guaranteed as hard constraint. Worker k is qualified when $\eta^k_j = 1$.	
	Hard constraint:	$x^k_{i,j} \leq \eta^k_j \ ,\forall i \in V^S, \forall j \in T, \forall k \in K$

matrix to define minimum qualified level of every skill. The second approach cannot measure skill level differences between a worker and a visit to be made. However, it compresses both proficiencies and minimum qualified level into a boolean parameter as explain above. This approach requires only one matrix to present data, i.e. workforce-visit matrix to define whether a worker is qualified to make a visit or not. The second approach may require less computational memory. Therefore, we apply the skill and qualification constraints from models MAP-TTC and HCS-PCD (more detail in Section 2.4.6).

2.3.7 Working Hours Limit Constraints

Working hours limit constraint is to define a maximum working duration of each worker. This constraint is usually implemented in the mid-term or long-term planning or an application where workers have flexible working time. The implementations of working hours limit constraint are presented in Table 2.10. From the table, the only model to have the working hours limit constraint

Table 2.10: working hours limit implemented by five mathematical models.

	Working hours limit
ODS-HHC	Duration between any two visits must less than the maximum working hours.
	Hard constraint: $(\tilde{a}_j^k + \delta_j) - \tilde{a}_i^k \leq h^k$
NB-TCS	None defined
VRS-TPS	None defined
MAP-TTC	None defined
HCS-PCD	None defined

is ODS-HHC. The constraint explicitly describes the total working duration as the longest differences between an arrival time of visit i, $\tilde{a}_i^k$, and the finish time of visit j, $(\tilde{a}_j^k + \delta_j)$. The time differences between two visits must less than the maximum working duration of a worker k.

The other four models do not implement this constraint by assuming the time horizon or the workforce time availabilities duration are equal to their working hours limit. Some cases may assume that workers do not have working hours limits but all visits are booked during the day time. Therefore, assignments which respect to the other constraint will satisfy this constraint automatically.

The working hours limit constraint is become necessary when the workforce time availability constraint has been treated as a soft condition (see also 2.3.8). A soft condition of the workforce availability constraint theoretically allows assignments to be made for the whole time horizon where the total working hours may exceed the working limit. Although, the constraint in the model ODS-HHC may not be efficient when a solution shows visits required at the beginning and at the end of the time horizon. For example, a worker k to make the first visit at 00:30 AM for a 5-hour task and the second visit at 20:00 PM for a 3-hour task where the working duration is in total of 22:30 hours. This example may appear in HHC problem where a patient requires a care worker to sleep in the patient house.

The alternative approach is to implement this constraint as a normal available resource limit constraint. That is a summation of working hours spent must less than the maximum working hours limit, expressed as:

$$\sum_{i \in V^S} \sum_{j \in T} \delta_j x^k_{i,j} \leq h^k, \forall k \in K$$

The constraint might add travel time $t_{i,j}$ if the travel duration is included as working time. We apply the alternative approach to our implementation (see Section 2.4.7).

2.3.8 Workforce Time Availability Constraints

The constraint is defined to guarantee that visiting durations are placed during a workforce shift. Each worker k has a shift defined by the earliest working time α^k_L and the latest working time α^k_U. Generally, all assignments must take place within that shift or working time. However, some cases might allow assignments outside working shift but these are less preferred.

Table 2.11 presents mathematical formulations implemented to tackle time availability constraints. We found that only ODS-HHC implements this constraint as a soft condition. To implement this requirement, ODS-HHC has additional variables η^k_L and η^k_U as actual earliest working time and latest working time respectively. Thus, arrival time must be within the duration $[\eta^k_L, \eta^k_U]$. It also introduces s^3_k and s^4_k as positive variables to measure difference between actual working shift and defined working shift. As such, $s^3_k \geq \alpha^k_L - \eta^k_L$ and $s^4_k = \eta^k_U - \alpha^k_U$. Additionally, overtime O^k is a duration that exceeds the allowed working limit h^k. Hence, the overtime can be calculated from $O^k \geq \eta^k_U - \eta^K_L - h^k$ where O^k is a variable to be minimised.

For NB-TCS, the availability constraint is included in the assignment constraint. It tackles the case as a hard constraint, i.e. a visit j to be allocated outside a worker shift has $\eta_j^k = 0$ to enforce the solver to select other workers who are available where $\eta_j^k = 1$. This formulation cannot apply directly to the time window problem because a visit time window might partially overlap with workforce unavailable period. Therefore, a deterministic η_j^k cannot be determined.

The other models consider this requirement as a hard constraint, where assigned time must be allocated only within shift duration. The constraint is simply expressed as

$$\alpha_L^k \leq \tilde{a}_j^k \leq \alpha_U^k - \delta_j$$

However, not all arrival times can be applied by this constraint. Let us consider the models VRS-TPS and HCS-PCD, an arrival time $\tilde{a}_j^k$ must be within a visit time window when an assignment is made $x_{i,j}^k = 1$ which result in $w_j^L \leq \tilde{a}_j^k \leq w_j^U$. By assuming $\alpha_L^k \leq w_j^L \leq w_j^U \leq \alpha_U^k$, we can see that $\tilde{a}_j^k$ is valid in both constraints. However, if a visit j is not assigned to a worker k, then $\tilde{a}_j^k = 0$. Thus, the value will contradict this constraint because $\tilde{a}_j^k$ may be less than α_L^k if $\alpha_L^k > 0$. To fix this issue, VRS-TPS and HCS-PCD only apply this constraint to arrival times at the depot only.

On the other hand, MAP-TTC uses integer variables $x_{i,j}^k$ to control this constraint. The constraint will active only when $x_{i,j}^k = 1$, otherwise the left hand side of the inequality will always less than a_j^k because M is a big constant value. In addition, it is sufficient to apply this constraint only for the start and the end nodes because visits in between will have visiting arrival time after the first visit but before the last visit.

The workforce time availability constraints act similarly as the visit time window in the way that visiting arrival time must be made within a certain time frame. In some cases, the workforce time availability constraint might con-

Table 2.11: Workforce time availability implemented by five mathematical models.

	Workforce time availability	
ODS-HHC	Hard and soft constraints for time availability and overtime duration which need to be minimised.	
	Soft constraint:	$\eta_L^k + t_{i,j} - M(1 - x_{i,j}^k) \leq \tilde{a}_j^k \; \forall j \in T, i \in D, \forall k \in K$ $\tilde{a}_j^k + t_{j,i} + \delta_j - M(1 - x_{j,i}^k) \leq \eta_U^k \; \forall j \in T, i \in D', \forall k \in K$ $s_k^3 \geq \alpha_L^k - \eta_L^k$ $s_k^4 \geq \eta_U^k - \alpha_U^k$
	Soft cons. obj.	$\sum_{k \in K} s_k^3 + s_k^4 \qquad s^3, s^4 \geq 0$
	Overtime const	$O^k \geq \eta_U^k - \eta_L^k - h^k$
	Overtime obj.	$\sum_{k \in K} O^k \quad , O^k \geq 0 \; \forall k \in K$
NB-TCS	See NB-TCS assignment constraint.	
VRS-TPS	Time from the starting node and ending node are within the time availability restriction.	
	Hard constraint:	$\alpha_L^k \leq \tilde{a}_j^k \leq \alpha_U^k - \delta_j \; \forall k \in K, \forall j \in D \cup D'$
MAP-TTC	First and last visits must lie between worker's time availability.	
	Hard constraint:	$\alpha_L^k + t_{0,j} - M(1 - x_{0,j}^k) \leq a_j^k$ $a_i^k + \delta_i + t_{i,0} - M(1 - x_{i,0}^k) \leq \alpha_U^k$
HCS-PCD	Last visit must finish in between workforce time availability.	
	Hard constraint:	$\alpha_L^k \leq \tilde{a}_0^k \leq \alpha_U^k - \delta_j$

flict with the visiting time window constraints, for example, a workforce available only in the afternoon but a visit required to be made during the morning. Therefore, either workforce time availability constraint or visiting time window constraint must be implemented as soft conditions to prevent constraint conflicts. The choice to be made depends on the nature of business. In some case, such as broadband companies, they might prefer to have soft conditions at visiting time window constraints rather than the workforce time availability constraint. On the other hand, a home healthcare business might prefer to soften the workforce time availability conditions because some visits are highly time dependent.

In our implementation, we apply the workforce time availability as a soft condition. We adapt the constraint in ODS-HHC by adding decision variables to add violation costs in the objective function (more detail in Section 2.4.8.

2.3.9 Special Cases: Time-dependent Constraints

The only special case we would like to discuss in this thesis is time-dependent constraints. They are formulations describing two related visits. Two visits can be related time-wise such as synchronised visits, overlapped visit, etc. Generally, the constraint defines limit on the time differences between two visits, e.g. synchronised visits have no time difference in terms that the two visits must start at the same time.

Only two of the models considered here have incorporated time-dependent constraints. VRS-TPS implements two sets of time-dependent constraint: synchronisation constraint and precedence constraint. Precedence constraint can cover two cases: overlapped visit and non overlapped visit.

HCS-PCD proposed generalised precedence constraints where a single formulation can cover five precedence conditions: synchronisation, overlapped, minimum difference, maximum difference, and min-max difference. This con-

Table 2.12: Problem specific constraint implemented on five mathematical models.

	Special constraints
ODS-HHC	Do not define
NB-TCS	Do not define
VRS-TPS	Synchronisation and precedence constraints
	Sync. constraint: $\sum_{k\in K} \tilde{a}_i^k = \sum_{k\in K} \tilde{a}_j^k \; \forall (i,j) \in S_{sync}$ S_{sync} is a set of synchronised visits. Members are pairs of visit (i,j) in which visit i and j must be attended synchronously.
	Prec. constraint: $\sum_{k\in K} \tilde{a}_i^k \leq g(i,j) + \sum_{k\in K} \tilde{a}_j^k \; \forall (i,j) \in S_{prec}$ S_{prec} is a set of precedence visits. Members are pairs of visit (i,j) in which visit i must be attended before visit j. $g(i,j) = -\delta_i$ workforce does not arrive a visit j before service of visit i. $g(i,j) = 0$ and $g(j,i) = \delta_i$ when additional visit j must be made during service of the first visit i.
MAP-TTC	Do not define
HCS-PCD	Generalised precedence constraints.
	Hard constraint: $w_i^L y_i + \sum_{k\in K} \tilde{a}_i^k + s_{i,j} \leq \sum_{k\in K} \tilde{a}_j^k + w_j^U y_j \; \forall (i,j) \in S, s_{i,j} \in \mathbb{R}$

straint implementation will be discussed further in chapter 6.

2.3.10 Home Healthcare Problem Requirements and Constraints in the Literature

From the five selected models, an individual set of constraints cannot cover the HHC problem that exists in our industrial scenarios. However, the integration of features from those models does. Table 2.13 summarises the HHC requirements, five recently mathematical models [13, 26, 55, 107, 121] shown earlier and a newly proposed MIP model where its detail is shown in Section 2.4. The table is presented in three sections. The first section presents types of mathematical model implemented to solve WSRP. The second part lists components of objective functions and presents models that support those objective. The

Table 2.13: Comparison of requirement from real world data, proposed model and others in the literature

	HHC dataset	ODS-HHC [121]	NB-TCS [13]	VRS-TPS [26]	MAP-TTC [55]	HCS-PCD [107]	Proposed model
Model type		MIP	IP	MIP	IP	IP	MIP
Objectives							
Distance	Tier1	-	-	-	-	Yes	Tier1
Visit Preference	Tier2	Yes	-	Yes	-	Yes	Tier2
Region Violation	Tier3	-	-	-	-	-	Tier3
Time Availability Violation	Tier3	-	-	-	-	-	Tier3
Unassigned visit	Tier4	-	-	-	-	Yes	Tier4
Min worker	-	-	Yes	-	Yes	-	-
Balance workload	-	-	Yes	Yes	-	-	-
Constraints							
Visit Assignment	Soft^1	Hard	Hard	Hard	Hard†	Soft^{1‡}	Soft^{1†‡}
Route Continuity	Hard	Hard†	Hard†	Hard†	Hard†	Hard†	Hard†
Start and End Locations	Multi	Single	-	Multi†	Single	Multi†	Multi†
Travel Time Feasibility	Hard	Hard†	-	Hard†	Hard†	Hard	Hard†
Time window	Hard	H/S	-	Hard	Hard†	Hard	Hard†
Skill and Qualification	Hard^2	Hard	Hard	-	Hard†	Hard†	Hard^{2†}
Working Hour Limit	Hard	Hard†	-	-	-	-	Hard†
Workforce Time Availability	Soft	Soft†	Hard	Hard	Hard	Hard	Soft†
Workforce Region Availability	Soft^2	-	-	-	-	-	Soft^2
Time-dependent	-	-	-	Yes	-	Yes	-

^{1} Unassigned visits are used as part of the soft conditions.
^{2} Variables related to the constraint are used in visit preference.
†,‡ Proposed model shares formulations.

third part shows lists of constraints to be implemented or presented amongst the models.

The requirements of HHC consist of a four-tier objective functions, six hard conditions and three soft conditions. The visit assignment constraint, where the implementation is required to be a soft condition, has variables related to unassigned visits where it is the tier 4 of the objective function to be minimised. The other two variables: out of region visit and out of working time visit are related to geographical region constraint and workforce time availability constraint, respectively. Both variables are allocated as tier 3 objective value. Visit preferences as presented in the tier 2 objective value considers three preference sources: additional skill preferences, geographical region preferences, and workforce-visit preferences. The tier 1 objective value is to minimise travel distances and the other monetary costs such as workforce salary where it could pay by visit hours.

Table 2.13 shows model ODS-HHC meets almost all HHC requirements apart from the geographical region constraint and the multiple skills and qualifications constraint. However, the proposed model uses mathematical formulation from model MAP-TTC (5 constraint types), followed by HCS-PCD (4 constraint types), and adopted 3 constraint types from ODS-HHC and VRS-TPS. This table shows clearly that the proposed MIP model is built based on the five selected models. However, there are small modifications to the formulation to adapt the constraint in response to the problem requirements. More detail on the development of the proposed MIP model is explained in the next section.

Apart from the five selected model above, it is worth mentioning that there is an integer programming model proposing to solve a home care problem where it considers a joint scheduling and routing problem Cappanera and Scutellá [33]. The integer programming model is targeted to solve the problem in a weekly planning horizon. This problem requires visiting patterns such as a patient should be visited on Monday, Wednesday and Friday. The pattern is defined by care plan which has been agreed prior the scheduling process. The integer programming model implemented in this work is to define a flow model. The model contains several constraints dedicated to define the flow of visits and to build visit patterns. In addition, this work also introduces *days of week* as additional dimension to the visit assignment variable x. However, there are some constraints omitted from this model which are workforce time availability constraint and time window constraint. Additionally, this work interprets travel time feasibility constraint in a different way where visits within one day must have a total travel time and visiting time less than a working duration of that worker. Therefore, the visit arrival times have not been produced. We acknowledge that this approach proposes a way to avoid the use of positive variables where they are usually required to define arrival times and time windows. Overall, this model is defined differently to the other models in the literature but does not implement some features.

2.4 Home Healthcare Scenarios and Implemented Model

This section presents mathematical formulations implemented as a MIP model to define HHC problems. The notation used in the MIP model is the same as presented in the previous section, which is listed in Table 2.3.

The concept of the model is to present the HHC problem by a graph $G = (V, E)$, the same principle with the other five selected models from the literature. The graph G composes of a set of nodes V and a set of edges E where each edge connects between two nodes. A node in V can be a visit, a start location, or an end location. Therefore, $V = D \cup T \cup D'$ where D is a set of start locations, T is a set of visits, and D' is a set of end locations. A directed edge in E represents a connection between two nodes, e.g. two visits, a visit and a start location, etc. For convenience, we define $V^S = D \cup T$ as nodes that have leaving edges and $V^N = D' \cup T$ as nodes that have entering edges. The HHC problem is to assign workers $k \in K$, where K is a set of workers, to directed edges links between two nodes. Edges link beginning from a start location, passing through visits, and terminating at an end location are form a working path. A worker must have at most one working path for the whole time horizon. Mathematical formulations to define the problem are presented, next.

2.4.1 Visit Assignment Constraint

This HHC problem requires the visit assignment constraint to be implemented as a soft condition where some visits may be left unassigned. The model makes decisions through binary decision variables $x^k_{i,j}$ where the variable is equal to 1 when a directed edge from node i to node j is assigned to worker k; otherwise, $x^k_{i,j} = 0$. Multiple workers, in total of r_j, are requested to make a single visit. By these two requirements, a soft condition implementation and a visit with multiple workers requirement, we integrate two constraints from the literature, MAP-TTC and HCS-PCD, to define this visit assignment constraint. The constraint is then formulated as:

$$\sum_{k \in K} \sum_{i \in V^S} x^k_{i,j} + y_j = r_j \qquad \forall j \in T \tag{2.1}$$

The integration of the two constraints forces y_j to be a positive integer variable instead of a binary variable. This is to accommodate when none of workers can make this visit, thus $y_j = r_j$ and the total number of unassigned visits is $\sum\limits_{j \in T} y_j$. In this case, we treat unassigned visit and assignment left unassigned indifferently.

2.4.2 Route Continuity Constraints

This model adopts the flow concept because it has been used by the five selected models. The flow conservation constraint is implemented in all the five selected models. Therefore, the formulation for HHC problem remains the same, which can be written as:

$$\sum_{i \in V^S} x^k_{i,j} = \sum_{n \in V^N} x^k_{j,n} \qquad \forall j \in T, \forall k \in K \tag{2.2}$$

That is, for each visit j and worker k, the total number of assigned entering edges (left hand side of the equation) must equal to the total number of assigned leaving edges (right hand side of the equation). This is to guarantee that every worker who makes this visit must leave after they finish their work in this visit.

2.4.3 Start and End Locations Constraint

These constraints are to define the start and end of a worker path. The HHC requirements is to have multiple start and end locations so that a worker can start their journey from home. The only implementation that could fit to the problem requirements is the constraints implemented in HCS-PCD model which require the summation of leaving edges from the start location equal to 1. The same equation also applies to the entering edges and the end location. The formulation are

$$\sum_{j \in V^N} x_{i,j}^k \leq 1 \qquad \forall i \in D, \forall k \in K \tag{2.3}$$

$$\sum_{i \in V^S} x_{i,j}^k \leq 1 \qquad \forall j \in D', \forall k \in K \tag{2.4}$$

A small modification is made by replacing $=$ with $\leq$ to allow the case that a worker k is not assigned to any visit where $x_{i,j}^k$ can be 0. The additional formulations are added to guarantee the left hand side of inequality (2.3) and (2.4) to be 1 when the worker k is assigned to make visits by the following constraints:

$$\sum_{j \in V^N} x_{n,j}^k \geq \sum_{j \in V^N} x_{i,j}^k \qquad \forall k \in K, \forall i \in T, \exists n \in D \tag{2.5}$$

$$\sum_{i \in V^S} x_{i,n}^k \geq \sum_{i \in V^S} x_{i,j}^k \qquad \forall k \in K, \forall j \in T, \exists n \in D' \tag{2.6}$$

The two constraints, (2.5) and (2.6), force the left hand side of the inequality to be 1 if a worker k is assigned to make at least one visit where the right hand side of the inequality is 1. Therefore, when considering all four constraints, the assignment from the start location must be made when a worker k has at least one visit to be made.

2.4.4 Travel Time Feasibility Constraint

This constraint defines an arrival time at visit j and eliminates assignment cycles. From the previous section, there are four models implementing this constraint and the formulations have the same structure. We choose the simpler version of inequality, presented in ODS-HHC, MAP-TTC, and HCS-PCD, to implement in our HHC model as:

$$a_j^k + M(1 - x_{i,j}^k) \geq a_i^k + \delta_i + t_{i,j} \qquad \forall k \in K, \forall i \in V^S, \forall j \in V^N \tag{2.7}$$

That is, if a worker k is assigned to make visit j after visit i, where $x_{i,j}^k = 1$, the arrival time at visit j (left hand side of inequality) must be at least a summation of the arrival time a_i^k at visit i, working duration δ_i at visit i, and travel time $t_{i,j}$ between visit i and visit j (right hand side of the inequality). We can see that the inequality is always valid if $x_{i,j}^k = 0$ and M is a large constant, which results in deactivating the boundary on arrival time.

2.4.5 Time Window Constraints

The HHC problem tackled here does not have time windows but require an exact arrival time. Generally, a time window problem enforces an arrival time of a worker to make a visit during a time period $[w_j^L, w_j^U]$. This concept also support an exact arrival time scenario by having $w_j^L = w_j^U$ where a_j^k must be exactly w_j^L. Therefore, we can use one of five time window constraints proposed in the five models in the literature. We choose a simple time window constraint in model MAP-TTC because the outcome of these implementations are indifferent and this formulation gives a cleaner look to the model. The formulation can be written as:

$$w_j^L \leq a_j^k \leq w_j^U \qquad \forall j \in T, \forall k \in K \tag{2.8}$$

2.4.6 Skill and Qualification Constraint

The HHC requirement is to assign qualified workers to make visits where each visit may request a variant of skills. Basically, this constraint reduces the size of search spaces by eliminating workers who are not qualified to make a certain visit from the decision making. There are two main approaches implemented in the five selected models: using direct information (ODS-HHC), or using qualification indicator (MAP-TTC and HCS-PCD). This HHC model applies the qualification indicator approach, where the formulation can be define as:

$$x_{i,j}^k \leq \eta_j^k \qquad \forall k \in K, \forall i \in V^S, \forall j \in T \tag{2.9}$$

We note that η_j^k is a pre-processed binary parameter to indicate that a worker k can take a visit j when its value is 1, and when the value is 0 the visit cannot be assigned. The given data from our industrial partner is in a form of visit-skill and worker-skill. The visit-skill defines a list of skills requested by a visit (also proficiency level of each skill) and the worker-skill defines a list of skills possessed by a worker (also with their proficiency level). The pre-processing runs through these two sets of values and returns η_j^k. From the constraint, the decision variable $x_{i,j}^k$ is allowed to be 1 only if $\eta_j^k = 1$.

2.4.7 Working Hour Limit Constraint

This condition is restricted to working hours. Hence, the summation of all visiting durations δ_j cannot exceed the allowance h^k for each worker k. This constraint is used by only. However, the constraint from might not be suitable for this HHC instances because the instances have visits distributed for the whole

24 hours time horizon. Therefore, we use an optional approach that implements this formulation in a simple form such that the summation of duration δ^j spent in every visit must less than the working hour limitation h^k, which is written as:

$$\sum_{i \in V^S} \sum_{j \in T} x^k_{i,j} \delta_j \leq h^k \qquad \forall k \in K \tag{2.10}$$

2.4.8 Workforce Time Availability Constraint

This constraint is implemented as a soft condition. The only soft condition implemented in the five selected models is a constraint in ODS-HHC where durations of visits outside worker's time availability are the cost of violation. However, the violation of our HHC problem is the total number of visits which are allocated outside worker's availability period. Therefore, we modify this constraint based on the constraint from model ODS-HHC which the modified constraint can be written as:

$$\alpha^k_L - a^k_j \leq M(1 - x^k_{i,j} + \omega_j) \qquad \forall k \in K, \forall i \in V^S, \forall j \in T \tag{2.11}$$

$$a^k_j + \delta_j - \alpha^k_U \leq M(1 - x^k_{i,j} + \omega_j) \qquad \forall k \in K, \forall i \in V^S, \forall j \in T \tag{2.12}$$

A binary variable $\omega_j = 0$ indicates that a visit j has been assigned to a worker who has time availability that covers the whole visit; otherwise, $\omega_j = 1$. We can see from the inequalities that the constraints are deactivated if $\omega_j = 1$, i.e. the left hand side of the inequality will be always less than the right hand side. Therefore, assignments outside worker availability period is possible. However, an $\omega = 1$ also means a soft constraint is violated where the total soft constraint violation must be minimised. Thus, ω_j is also added to the objective function.

2.4.9 Region Availability Constraint

This HHC problem allows workers to choose their primary working regions. Normally, workers should be assigned to make visits in their working region. However, assignments made outside working regions are also applicable as long as the worker is eligible to take it. The implementation of this constraint is tailor-made because none of five selected works considers this requirement.

We may use the same formulation for skills and qualifications constraint with a small modification as:

$$\sum_{i \in V^S} x_{i,j}^k - \psi_j \leq \gamma_j^k \qquad \forall k \in K, \forall j \in T \tag{2.13}$$

We introduce the binary parameter $\gamma_j^k = 1$ to indicate that visit j is located in the available region of worker k. Thus, assigning worker k to visit j can be made without penalty. The binary variable $\psi_j = 0$ is a variable to indicate that the assignment made at visit j is located within worker's availability region; otherwise $\psi_j = 1$. For the case that $\gamma_j^k = 0$, assigning variable $x_{i,j}^k = 1$ is feasible only when binary variable $\psi_j = 1$, where there is a worker who is not available in the visiting region. However, the assignment can be made with a soft constraint penalty.

2.4.10 Objective Function

The objective function (2.14) to be minimised involves three costs: monetary cost, preferences penalty cost, soft constraints penalty cost, and unassigned visits. This objective function has been defined in consultation with our industrial partners as it seeks to incorporate the key aspects that make a high-quality solution, low operational cost and improved satisfaction of patients and workers. Those costs are balanced into four tiers, each corresponding to the weight parameters $\lambda_1, ..., \lambda_4$. The designing of multi-tier cost function basically defines the order of importance of operational cost. Hence, weights that are set as the higher rank tier have a significantly higher value than the lower rank one [35, 107].

$$\text{Min} \sum_{k \in K} \sum_{i \in V^S} \sum_{j \in V^N} \lambda_1 \left(d_{i,j} + p_j^k \right) x_{i,j}^k + \sum_{k \in K} \sum_{i \in V^S} \sum_{j \in V^N} \lambda_2 \rho_j^k x_{i,j}^k + \sum_{j \in T} (\lambda_3(\omega_j + \psi_j) + \lambda_4 y_j) \quad (2.14)$$

Unassigned visit is the first priority of this problem because the business requires to maximise delivery of services. The weight λ_4 which is correspondent with unassigned visit is set as the highest value. The number of unassigned visits is the summation of y_j for all $j \in T$.

Soft constraint violation is the second priority. The soft constraint violation is shown in two constraints: region availability constraint and time availability constraint. The priority for those two constraint types is the same. Weight parameter for soft constraint violation is λ_3 set to be a large number but smaller than λ_4.

Preference penalties are the third priority as they are required by service providers. In these scenarios, customer experience is valued higher than the operational costs. Therefore, weight parameter λ_2 is set to be less than λ_3. Preference penalty is a cost charged when the service provider could not fulfill the highest preference level.

The value of preferences ρ_j^k when assigning a worker k to visit j ranges from 0 to 3 where 0 means no penalty charged and 3 is full penalty. This is because of the satisfaction of the three types of preferences: additional skill requirement, workforce region preference, and workforce-client relationship. For each assignment has a value in the range $[0, 1]$ from satisfied to not satisfied.

Monetary costs are payments made for providing services. Monetary costs include travelling costs, salaries, and equipments used to deliver a visit. For this scenario, the monetary costs are the lowest rank. The weight correspondent with this part is λ_1 which is set to be minimum priority.

With multi-tier objective implemented, we have seen that the geographical region involved two cost tiers: region violation penalty and worker region preference penalty. The first part, region violation penalty, basically counts assignments made outside the predefined available regions. This is considered as soft constraint violation. For example, from constraint (2.13) if $\gamma_j^k = 0$ and $x_{i,j}^k = 1$, then $\psi_j = 1$ which adds $\lambda_3\psi_j$ to the objective function. On the other hand, if the assignment allocates the worker to his available regions ($\gamma_j^k = 1$), then it involves the preference penalty cost (ρ_j^k) where a worker may be assigned to less preferred regions. This charges objective function in total by $\lambda_2\rho_j^k$. Note that, with the tier weight scheme, an assignment with preference penalty always gives less cost than the one with region violation.

The similar practice is also implemented to skills and skill requirements. A visit may require a set of base skills and additional skills. The base skills are affected by constraint (2.9) such that if a worker does not have base skills for the visit, they cannot be assigned. If the worker is compatible according to base skills, then skill penalty costs take place when he cannot fulfil additional skills. It is worth mentioning that all parameters are given together with data instances.

From the above constraints and objective function, we have the final MIP model for the HHC problem, which is solved using the methods in Chapter 3, 4, 5, and 7. The final MIP model is presented as:

$$\text{Min} \sum_{k\in K}\sum_{i\in V^S}\sum_{j\in V^N} \lambda_1 \left(d_{i,j} + p_j^k\right) x_{i,j}^k + \sum_{k\in K}\sum_{i\in V^S}\sum_{j\in V^N} \lambda_2 \rho_j^k x_{i,j}^k + \sum_{j\in T} (\lambda_3(\omega_j + \psi_j) + \lambda_4 y_j)$$

$$\begin{aligned}
\text{Subject to} \quad \sum_{k\in K}\sum_{i\in V^S} x_{i,j}^k + y_j = r_j & \qquad \forall j \in T \\
\sum_{i\in V^S} x_{i,j}^k = \sum_{n\in V^N} x_{j,n}^k & \qquad \forall j \in T, \forall k \in K \\
\sum_{j\in V^N} x_{n,j}^k \geq \sum_{j\in V^N} x_{i,j}^k & \qquad \forall k \in K, \forall i \in T, \exists n \in D \\
\sum_{i\in V^S} x_{i,n}^k \geq \sum_{i\in V^S} x_{i,j}^k & \qquad \forall k \in K, \forall j \in T, \exists n \in D' \\
\sum_{j\in V^N} x_{i,j}^k \leq 1 & \qquad \forall i \in D, \forall k \in K \\
\sum_{i\in V^S} x_{i,j}^k \leq 1 & \qquad \forall j \in D', \forall k \in K \\
x_{i,j}^k \leq \eta_j^k & \qquad \forall k \in K, \forall i \in V^S, \forall j \in T \\
a_j^k + M(1 - x_{i,j}^k) \geq a_i^k + x_{i,j}^k t_{i,j} + \delta_i & \qquad \forall k \in K, \forall i \in V^S, \forall j \in V^N \\
w_j^L \leq a_j^k \leq w_j^U & \qquad \forall j \in T, \forall k \in K \\
\sum_{i\in V^S}\sum_{j\in T} x_{i,j}^k \delta_j \leq h^k & \qquad \forall k \in K \\
\sum_{i\in V^S} x_{i,j}^k - \psi_j \leq \gamma_j^k & \qquad \forall k \in K, \forall j \in T \\
\alpha_L^k - a_j^k \leq M(1 - x_{i,j}^k + \omega_j) & \qquad \forall k \in K, \forall i \in V^S, \forall j \in T \\
a_j^k + \delta_j - \alpha_U^k \leq M(1 - x_{i,j}^k + \omega_j) & \qquad \forall k \in K, \forall i \in V^S, \forall j \in T \\
x_{i,j}^k, \gamma_j, \omega_j, \psi_j \text{ are binary}, y_j \text{ are integer}, a_j^k \geq 0 & \qquad \forall i, j \in T, \forall k \in K
\end{aligned}$$

2.5 Sets of Problem Instances

The problem instances used here have been provided by our industrial partner. There are 42 instances that make available to the public.[1] Their operation is to deliver workforce management software to several home healthcare service providers. The following are the data specification.

- A visit is located in one region, a region may have multiple visits;
- A worker has available regions, no penalties to make visits;
- A worker makes visits outside their regions with penalty costs, this should be avoid as much as possible;
- A worker has one available time horizon;
- A worker can make visits outside their available time, this should be avoid as much as possible;
- A visit has minimum skill requirements and additional skill requirements with prefer level [0,1] (1 is the most prefer);
- A visit may have preferred workers with prefer level [0,1] (1 is the most prefer);
- Only 1 transportation mode available, assumed car;
- Distances and travel times between places are Euclidean;
- Distance and travel time matrices are both symmetric;
- Travel times are proportional to distances (Time = 1.5 Distance);
- Monetary costs are differed by visits and by workers.

[1]`https://drive.google.com/open?id=0B2OtHr1VocuSNGVOT2VSYmp6a2M`.

The data originated from six distinct home healthcare companies, named here as sets A, B, C, D, E, and F which are ordered from small to large. Each set has seven instances which were randomly selected from different periods. A problem instance presents a one day planning operation.

Table 2.14 presents basic information about the 42 instances. The information includes a number of visits, a number of workers, a number of regions, average visit duration (in minutes), percentage of time with maximum simultaneous visits for the whole time horizon, and overall compatibility. The overall compatibility is provided by the average number of skilled workers that have time and region availability to perform a visit. Problem size can be simplified by the number of visits and the number of workers.

The maximum simultaneous visits highlight the minimum number of workers to be deployed, e.g. C-06 has 94.9% Max. simultaneous visits means 150 out of 158 visits are overlapped so the plan should deploy at least 150 workers simultaneously. We can see that instance set C has highest Max. simultaneous visits (66.6% - 94.9%).Most of the instance in sets A, B, and E have max. simultaneous visit between 20%-40%, except C-04 (16.6%), E-04 (17.3%), and C-06 (17.9%). The instances in set D and F have lower max. simultaneous visits than 20% with a minimum of 14.1%.

The overall compatibility presents an average number of workers that can be assigned to a visit. For example, A-07 which has compatibility at 1.2 shows that most visits have only one worker to choose and if all visits are assigned, that solution should be very near optimal solution because there are not many permutations to assign workers. The instances with high overall compatibility have more flexibility to make assignments, such as E-01 which can choose one of of 85.5 workers to make a visit. All instances in set E have very high compatibility score (69.2 - 95.3) comparing to the rest (1.2 - 20.3).

Table 2.14: Information on the WSRP instances obtained from real-world operational scenarios.

	Set A							Set B							Set C						
	01	02	03	04	05	06	07	01	02	03	04	05	06	07	01	02	03	04	05	06	07
# Visits	31	31	38	28	13	28	13	36	12	69	30	61	57	61	177	7	150	32	29	158	6
# Workers	23	22	22	19	19	21	21	25	25	34	34	32	32	32	1037	618	1077	979	821	816	349
# Regions	6	4	5	4*	4*	8*	4*	6	5*	7*	5*	8	8*	7	8	4	7	8	6	11	6*
Avg. Visit Duration	144	110	102	70	57	157	104	160	155	112	58	106	101	103	440	599	456	356	344	469	484
Max Simul. Visits (%)	32.2	35.4	31.5	25.0	30.7	35.7	30.7	36.1	33.3	24.6	16.6	22.9	29.8	22.9	90.4	85.7	92.6	75.0	75.8	94.9	66.6
Compat	6.4	6.1	5.5	7.2	2.4	5.8	1.2	17.6	6.9	19.5	6.3	14.7	16.3	15.4	10.6	3.1	6.7	10.4	3.8	7	1.2

	Set D							Set E							Set F						
	01	02	03	04	05	06	07	01	02	03	04	05	06	07	01	02	03	04	05	06	07
# Visits	483	454	585	520	538	610	611	418	425	462	351	461	301	498	1211	1243	1479	1448	1599	1582	1726
# Workers	164	166	174	174	173	174	173	243	244	267	266	278	278	302	805	769	898	789	889	783	1011
# Regions	13*	12*	15*	15*	15*	15*	14*	13*	14*	15*	13*	15*	13*	16*	45	46	54	47*	59*	44*	64
Avg. Visit Duration	62	50	67	61	58	57	56	99	102	101	73	100	64	98	79	87	83	89	72	81	92
Max Simul. Visits (%)	16.5	17.1	17.2	16.3	16.9	16.5	15.5	23.2	24.0	25.1	17.3	23.6	17.9	24.7	18.4	19.7	18.1	16.1	16.5	14.1	19.2
Compat	11.3	9.3	15.9	10.9	15.6	18.9	20.3	85.5	88.2	90.1	70.9	92.4	69.2	95.3	7.7	7.3	9.7	8.8	9.4	9.2	10.4

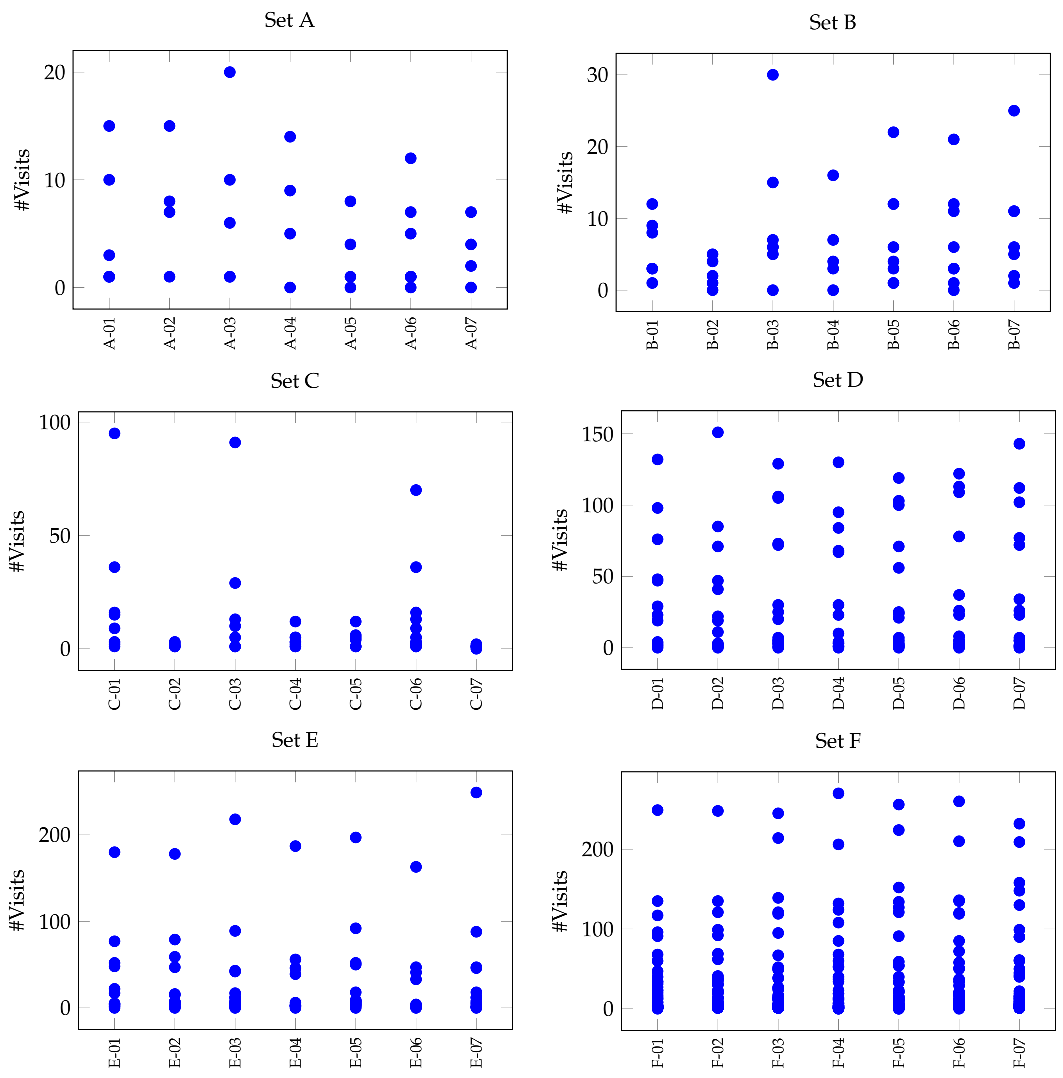


Figure 2.1: Scatter plots present distribution of the number of visits in geographical regions over a problem instances. Plots are presented in six sub-figures, each represents a problem scenario. Each scenario has seven problem instances, presented in X-axis in a sub-figure. Y-axis presents a number of visits. Each dot represent a geographical region.

Figure 2.1 presents the number of demanding visits demanded for each region of the 42 problem instances. The plot is grouped into four sub-figures, each represents a problem set. Each sub-figure has seven instances plotted on the X-axis. The Y-axis shows the number of visits. Each dot represents a geographical region in a problem instance. The figure shows that the number of visits are not balanced across geographical regions. From the figure, set C presents interesting cases, in C-01, C-03 and C-06, where demands in regions are usually less than 50 visits except only one region that has higher demand while region demands in other instances are group in very narrow ranges.

2.6 Mixed Integer Programming to Solve Home Healthcare Problems

We implemented the MIP model presented above to tackle home health care scenarios. We use an MIP solver, IBM ILOG CPLEX Optimization studio 12.4 [1], to solve the MIP problem. The solver runs on Windows 7 system with Intel Core i7-3820 CPU processor and 16 GB of RAM.

2.6.1 Exact Method to Solve Home Healthcare Instances

This part presents a result of using the MIP solver to solve real-world HHC instances. Table 2.15 shows objective values (in Fitness columns) and computational times (in Time columns) provided by MIP solver to solve 42 instances. Only 18 instances can be solved to optimality. Amongst the 18 instances, there are 3 instances in which the MIP solver can find optimal solution within 1 second. The longest computational time is 6,003 seconds when the MIP solver tackles B-03. The MIP solver ran out of memory on 24 instances where they are labeled *N/K* in the table.

Table 2.15: Objective value and computational time of 42 test instances using MIP solver.

Set	Fitness	Time(s)	Set	Fitness	Time(s)	Set	Fitness	Time(s)
A-01	3.49	7	B-01	1.70	21	C-01	N/K	N/K
A-02	2.49	8	B-02	1.75	2	C-02	3.15	6
A-03	3.00	14	B-03	1.72	6003	C-03	N/K	N/K
A-04	1.42	5	B-04	2.07	25	C-04	11.15	90
A-05	2.42	1	B-05	1.82	585	C-05	12.34	55
A-06	3.55	5	B-06	1.62	184	C-06	N/K	N/K
A-07	3.71	1	B-07	1.79	300	C-07	4.30	1
D-01	N/K	N/K	E-01	N/K	N/K	F-01	N/K	N/K
D-02	N/K	N/K	E-02	N/K	N/K	F-02	N/K	N/K
D-03	N/K	N/K	E-03	N/K	N/K	F-03	N/K	N/K
D-04	N/K	N/K	E-04	N/K	N/K	F-04	N/K	N/K
D-05	N/K	N/K	E-05	N/K	N/K	F-05	N/K	N/K
D-06	N/K	N/K	E-06	N/K	N/K	F-06	N/K	N/K
D-07	N/K	N/K	E-07	N/K	N/K	F-07	N/K	N/K

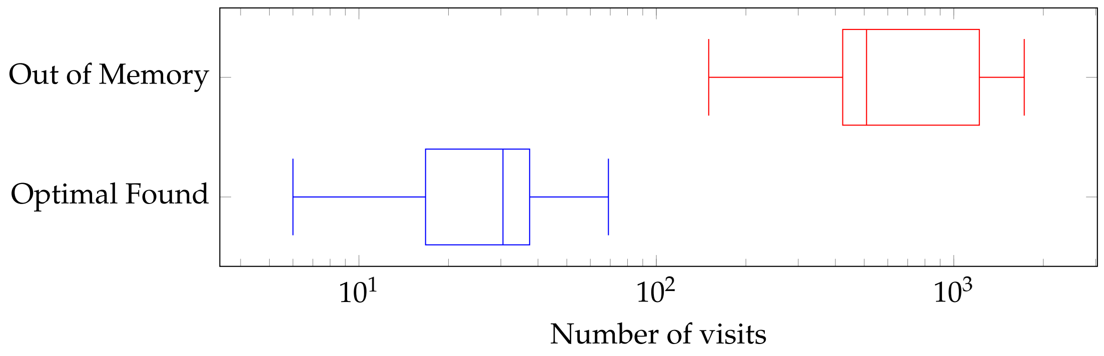


Figure 2.2: Box plot presents problem size distribution group by result of MIP solver.

The main reason that the MIP solver ran out of memory on 24 instances is that their problem size is too large to solve as a whole problem. The problem size can be estimated by the number of visits. Figure 2.2 presents distributions of the number of visits on instances grouped by results of MIP solver: optimal found and out of memory. The number of visits distribution on instances where the MIP solver can find the optimal solutions is presented in the blue box plot (below). This group has the number of visits which ranges from 6 to 69 visits, averagely 30.5 visits. The distribution of the number of visits on instances where the MIP solver ran out of memory is presented in the red box plot (above). The number of visits ranges from 150 to 1,726 visits, and 728.75

visits on average. It is clearly that the MIP solver is unable to process a problem where the number of visits is larger than 150 visits. However, the MIP performance remains unknown in the instances having the number of visits between 70 to 150 visits because we do not have real-world HHC instances where their size is in between this range.

The amount of memory that MIP solver requires is calculated by the number of constraints generated by the MIP model. Most of the constraints involve the number of visits and the number of workers denoted by $|T|$ and $|K|$ respectively. For example, the model generates the visit assignment constraint (2.1) equal to the number of visits $|T|$ (observing the number of generated constraints at formulation suffix, i.e. $\forall j \in T$), and the number of route continuity constraints (2.2) is the number of visits multiply by the number of workers, $|T||K|$. Note that we assume $|D| = |D'| = |K|$. Therefore, the estimated number of constraints to be generated by the MIP model can be calculated by:

$$\#\text{cons} = |K|^3 + 4|K||T|^2 + 7|T||K|^2 + 2|K|^2 + 3|K||T| + |K| + |T| \qquad (2.15)$$

From CPLEX guidelines, the amount of memory required by the MIP solver can be estimated as 1 MB per 1,000 constraints to load a problem instance to solve a linear programming problem [2]. Since the computer we used has 16 GB of RAM, the problem the MIP solver can tackle without running out of memory must have less than 16,384,000 constraints.

Table 2.16 shows the number of constraints generated and the computational memory estimation required for each of the 42 instances. The table is split into six blocks, for six sets of instances. Based of the number of constraints, the smallest instance is A-05, which has 54,049 constraints and its estimated memory requirement is 50 MB. The largest instance that can be solved practically is C-04 which required 5.78 GB of RAM. Meanwhile, the smallest instance

Table 2.16: Estimated amount of constraints and memory requirement of 42 test instances.

Set	#cons.	GB	Set	#cons.	GB	Set	#cons.	GB
A-01	218,623	0.21	B-01	306,736	0.30	C-01	1.41 e08	138
A-02	203,311	0.20	B-02	84,712	0.08	C-02	270,691	0.26
A-03	270,000	0.26	B-03	1,254,601	1.22	C-03	1.05 e08	103
A-04	139,564	0.14	B-04	409,900	0.40	C-04	5,937,667	5.78
A-05	54,049	0.05	B-05	954,301	0.93	C-05	3,873,507	3.77
A-06	164,248	0.16	B-06	864,825	0.84	C-06	9.19 e07	90
A-07	65,323	0.06	B-07	954,301	0.93	C-07	142,398	0.14
D-01	2.49 e08	249	E-01	3.57 e08	349	F-01	1.07 e10	10,488
D-02	2.29 e08	229	E-02	3.68 e08	360	F-02	1.04 e10	10,113
D-03	3.68 e08	368	E-03	4.78 e08	467	F-03	1.69 e10	16,537
D-04	3.04 e08	304	E-04	3.24 e08	316	F-04	1.34 e10	13,107
D-05	3.18 e08	318	E-05	5.08 e08	496	F-05	1.85 e10	18,018
D-06	3.94 e08	394	E-06	2.85 e08	279	F-06	1.51 e10	14,757
D-07	3.92 e08	392	E-07	6.46 e08	630	F-07	2.54 e10	24,839

that MIP solver ran out of memory is C-06 which required at least 90 GB of RAM. We ran the scenario using an HPC which has 100 GB of RAM but the HPC ran out of memory when solving relaxation problem. There is a big difference in the memory required between C-04 and C-06, approximately 84.2 GB. This value shows that upgrading computer memory will not be an efficient way. Furthermore, the largest HHC instance is F-07 in which the number of constraints is 2.54×10^{10}, where its estimated memory requirement is 24.8 TB. This value is an indication of the need to find alternative approaches to solve real-world HHC problem.

2.7 Summary

This chapter reviews WSRP constraints presented in the literature. Five mathematical programming models are selected and the formulation of each constraint type is presented. The five models are role models for the home health care scenarios that arose from our industrial partner. The proposed model is formulated in a mixed integer programming model where its constraints support all real-world requirements. The model objective function is to minimise the total operational costs which are the weighted sum of four types: cost of unassigned visits, cost of soft constraint violations, cost of preference satisfaction penalties, and monetary cost. A feasible solution must satisfy all MIP constraints. The constraints are implemented by adapting constraints from the literature and they are assembled to form a MIP model to solve HHC instances. There are 13 equations to explain 9 constraint types. Most constraints are implemented as hard conditions. Only visit assignment constraint, time availability constraint and region availability constraint are implemented as soft conditions.

The commercial MIP solver tackles the HHC instances implemented in the proposed MIP model. The result shows the MIP solver can find an optimal solution for 18 instances. These instances have the number of visits less than 150. The estimated memory requirements of these 18 instances are within 16 GB of RAM which is the amount set as maximum amount of RAM for our standard PC. The other 34 instances where the number of visits is more than 150 visits cannot be solved by the MIP solver because the instances require very high computational memory; at least 90 GB of RAM. This study also reveals that the solver may require up to 24.8 TB of RAM to load the largest instance where this amount of RAM is impossible to install in any standard PC.

The MIP solver has set benchmark results for 18 small instances for this research. However, there are memory limitations when using the exact approach to solve the larger real-world HHC instances. We can see that memory management is very crucial to use the MIP solver to find solutions efficiently. Therefore, the next chapters propose algorithms to find a feasible solution while harnessing the power of the MIP solver with a limited amount of computational memory.

Chapter 3

Traditional Decomposition for Home Healthcare Problem and Literature Review on Heuristic Decomposition Approaches

The previous chapter explains the MIP formulation for HHC. However, tackling the real-world instances with an MIP solver becomes difficult because the problem requires high computational resources. A decomposition method is one technique to deal with large scale problems. In this chapter we describe two main streams of the decomposition method: traditional decomposition and heuristic decomposition.

We made an attempt to solve the HHC problem using a traditional decomposition technique, a Dantzig-Wolfe decomposition. The results show the traditional decomposition technique cannot finish the process within 2 hours time limit. Thus, we survey heuristic decomposition methods. Section 3.2.1 shows that decomposing a problem into sub-problems with approximately equally number of visits result in faster computational times. These partitioning techniques are implemented for our problem in Chapter 5. We also survey methods in Section 3.2.2 and Section 3.2.3 that apply clustering techniques to partition problems. This will also later be used as part of our proposed method in Chapter 5. The method in Section 3.2.3 also lead us to reformulate the problem which will be the method in Chapter 7.

Traditional decomposition methods tackle a large scale problem in which the optimal solution can be guaranteed when the lower bound and upper bound match. These two bounds are calculated in every iteration. Traditional decomposition is commonly used in a linear programming problem (LP). Two main decomposition methods are the Dantzig-Wolfe decomposition [50] and Benders' decomposition [17].

Dantzig-Wolfe decomposition is a method for decomposing structured linear programming problems which was proposed by Dantzig and Wolfe [50]. The decomposition is used as a part of column generation algorithm [124] and branch-and-price algorithm [11, 113, 123]. Both algorithms were implemented to solve large scale LP/MIP such as a cutting stock problem [97], school timetabling problem [101], inventory and time constrained ship routing problem [39], multi-commodity network problem [10, 12], and vehicle routing problem [54, 111].

Benders' decomposition is also a technique to deal with very large linear programming problems [17]. This decomposition approach also requires a special block structure. The decomposition partitions a problem into two subproblems: a master problem and an auxiliary problem [47]. The master problem is a relaxed version of the original problem which contains only a subset of variables and the associated constraints. Solving a relaxed problem provides a lower bound to the original problem. The auxiliary problem has the same formulations as the full model but the variables related to the master problem are fixed as parameters. Solving an auxiliary problem provides an upper bound to the solution. The two sub-problems are solved iteratively until the upper and lower bounds are close. Benders' decomposition has been applied to real-world problems such as network design problems [20, 106], multi-commodity flow networks problem [66], distribution system planning [16], locomotive assignment [44, 129], aircraft routing and scheduling [45, 92, 93], vehicle routing problem [62], etc.

The rest of this chapter will focus only on the Dantzig-Wolfe decomposition implemented in the column generation algorithm because the Dantzig-Wolfe decomposition and the Benders' decomposition are related, i.e. the Benders' master problem is a dual problem of the Dantzig-Wolfe master problem in the linear programming problem [89].

3.1 Dantzig-Wolfe Decomposition Method in Column Generation Algorithm

We implement a column generation algorithm to solve HHC scenarios in order to study how traditional decomposition methods could work on the real-world HHC problem. This section starts by explaining the general concept of Dantzig-Wolfe decomposition, then continuing with the column generation implemented to solve the HHC problem, and finishing with an explanation of the computational result from the column generation algorithm.

3.1.1 Dantzig-Wolfe Decomposition for Linear Program

Dantzig-Wolfe decomposition requires a special LP structure known as *block-angular*. Given c and b are coefficient vectors, A is a coefficient matrix, x is a variable vector, and $(.)^T$ represents matrix transpose. A general LP can be written as:

$$\begin{aligned}\text{Minimise } \; & c^T x \\ Ax &= b \\ x &\geq 0.\end{aligned}$$

A sparse matrix A of the LP problem can be rearranged into coupling constraints and sub-matrices. We define $B_0, \ldots, B_k$ to be coefficient matrices of the coupling constraints, $A_1, \ldots, A_k$ to be coefficient matrices of sub-problems of A, b_0 is a right hand side coefficient vector of coupling constraints, $b_1, \ldots, b_k$ are right hand side coefficient vectors of sub-problems of A, and $x_0, \ldots, x_k$ are variable vectors. The block-angular structure is written as:

$$Ax = \begin{bmatrix} B_0 & B_1 & B_2 & \cdots & B_K \\ & A_1 & & & \\ & & A_2 & & \\ & & & \ddots & \\ & & & & A_K \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_K \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_K \end{bmatrix}$$

where coupling constraints presented in the top row sub-matrices are

$$\sum_{k=0}^{K} B_k x_k = b_0.$$

The idea of the Dantzig-Wolfe decomposition is to decompose the problem, such that a problem does not have to be solved with all sub-problems $A_k x_k = b_k$ included [123]. Instead, it solves only the coupling constraints called *master problem*. The rest of the matrix A are decomposed into sub-problems such that $A_k x_k = b_k$, for $k = 1, \ldots, K$, known as *pricing problem* which are then solved individually.

Master Problem

A main focus of Dantzig-Wolfe decomposition is the conceptualisation of the master problem. From the main problem, the master problem can be explained by:

$$\begin{aligned} \text{Minimise} \quad & \sum_k c^T x_k \\ & \sum_k B_k x_k = b_0 \\ & x \geq 0. \end{aligned}$$

However, we can write any point x as a linear combination of extreme points and extreme rays x^j of a problem P by *Minkowski's Representation Theorem* [99]. ν_j is a variable vector associated to each of x^j. Therefore, we can write decision variables by

$$\begin{aligned} x &= \sum_j \nu_j x^j \\ \sum_j \mu_j \nu_j &= 1 \\ \nu_j &\geq 0 \end{aligned}$$

where

$$\mu_j = \begin{cases} 1 \quad \text{if} \quad x^j & \text{is an extreme point} \\ 0 \quad \text{if} \quad x^j & \text{is an extreme ray} \end{cases}$$

We substitute the above formulations into the master problem assuming that we are dealing with K sub-problems, which results in:

$$\min \quad c_0^T x_0 + \sum_{k=1}^{K} \sum_{j=1}^{p_k} (c_k^T x_k^j) \nu_{k,j} \tag{3.1}$$

$$B_0 x_0 + \sum_{k=1}^{K} \sum_{j=1}^{p_k} (B_k x_k^j) \nu_{k,j} = b_0 \tag{3.2}$$

$$\sum_{j=1}^{p_k} \mu_{k,j} \nu_{k,j} = 1 \quad , \text{for} k = 1, \ldots, K \tag{3.3}$$

$$x \geq 0 \tag{3.4}$$

$$\nu_{k,j} \geq 0 \tag{3.5}$$

This master problem is a large linear programming problem because the number of extreme points and rays of each sub-problem is very large. This leads to very high number of variables $\nu_{k,j}$. We know that most of these variables will be non-basic variables, which do not require consideration within the problem at the problem initialization step. Therefore, only variables with a negative reduced cost will be added to the master problem, which is known as a *delayed column generation*. So the new formulation, called *restricted master problem*, can be written as:

$$\text{Minimise} \quad c_0^T x_0 + c^T \nu' \tag{3.6}$$

$$B_0 x_0 + B\nu' = b_0 \tag{3.7}$$

$$\sum \delta \nu' = 1 \tag{3.8}$$

$$x_0 \geq 0 \tag{3.9}$$

$$\nu \geq 0. \tag{3.10}$$

Note that the size of ν' is not fixed such that more variables will be added at every iteration of the algorithm.

Pricing Problem

We select the adding variable ν by its reduced cost. Given that π_1 is a dual variable of the constraint (3.7) of the restricted master problem and π_2^k is a dual variable of the constraint (3.8) of the restricted master problem, the reduced cost ($\sigma_{k,j}$) of the restricted master problem can be written as:

$$\sigma_{k,j} = (c_k^T x_k^j) - \pi^T B_k x_k^j = (c_k^T - \pi_1^T B_k) x_k^j - \pi_2^k \delta_{k,j}$$

The selected basic feasible solution x_k to enter the master problem must have minimum reduced cost. The sub-problem called *Pricing problem* can be written as:

$$\text{Minimise} \quad x_k \sigma_k = (c_k^T - \pi_1^T B_k) x_k - \pi_2^k \tag{3.11}$$

$$B_k x_k = b_k \tag{3.12}$$

$$x_k \geq 0 \tag{3.13}$$

The new columns x_k to enter the master problem must have $\sigma_k^* < 0$. The cost coefficient with respect to the entering column is $c_k^T x_k^*$. This can be solved by a linear programming solver.

3.1.2 Column Generation to Solve Home Healthcare Problem

We implement a column generation algorithm with small adaptations to solve the HHC problem presented in Chapter 2.

Master Problem

Table 3.1 presents the notation related to a master problem. Generally, the master problem chooses routes represented as columns. The main constraints are that a worker must have no more than one selected route and a task must be

Table 3.1: Additional notation used for the HHC master problem.

Symbol	Definition
Sets	
K	Set of workers.
T	Set of visits.
$j \in J_k$	Set of routes for worker k.
Parameters	
$c_j^k \in \mathbb{N}$	Cost of route j for worker k.
$\bar{x}_{tj}^k \in \{0,1\}$	Parameter indicating whether task t is assigned in route j for worker k.
λ_4	Cost of unassigned visits.
Variables	
y_i	Binary variable indicating visit i is unassigned.
ν_j^k	Binary variable indicating whether route j is chosen for worker k.

visited or penalised if it is unassigned. The objective is to minimise the overall cost of selected routes and penalty for unassigned visits.

$$\textbf{Minimise} \quad Z = \sum_{k \in K} \sum_{j \in J_k} c_j^k \nu_j^k + \sum_{i \in T} \lambda_4 y_i \tag{3.14}$$

subject to

$$\sum_{j \in J_k} \nu_j^k \leq 1 \qquad \forall k \in K \tag{3.15}$$

$$\sum_{k \in K} \sum_{j \in J_k} \bar{x}_{i,j}^k \nu_j^k + y_i = 1 \qquad \forall i \in T \tag{3.16}$$

$$\nu_j^k \in \{0,1\} \qquad \forall k \in K, j \in J_k \tag{3.17}$$

$$y_i \in \{0,1\} \qquad i \in T \tag{3.18}$$

The objective function (3.14) minimises the cost associated to each worker route, as well as the cost of unassigned tasks. Constraint set (3.15) ensures that at most one route is selected for each worker. Constraint set (3.16) ensures that a visit is either attended by a worker route or remains unassigned and then identified by the y_i variable.

Pricing Problem

Table 3.2: Additional notation used for the pricing problem in the HHC model.

Symbol	Definition
Parameters	
π^k	Dual variable values associated to the constraint set (3.15).
σ_j	Dual variable values associated to the constraint set (3.16).

$$
\mathcal{P}^k = \left\{
\begin{array}{lll}
\textbf{Minimise } R^k = \sum_{i \in V^S} \sum_{j \in T} \left(\lambda_1 (d_{i,j} + p_j^k) + \lambda_2 \rho_j^k \right) x_{i,j}^k & & \\
\quad + \sum_{j \in T} \lambda_3 (\omega_j + \psi_j) x_{i,j}^k & & \\
\quad - \sum_{i \in V^S} \sum_{j \in T} \sigma_j x_{i,j}^k - \pi^k & & (3.19) \\
\textbf{Subject to} & & \\
\sum_{k \in K} \sum_{i \in V^S} x_{i,j}^k \leq 1 & \forall j \in T & (3.20) \\
\sum_{i \in V^S} x_{i,j}^k = \sum_{n \in V^N} x_{j,n}^k & \forall j \in T & (3.21) \\
w_j^L \leq a_j^k \leq w_j^U & \forall j \in T & (3.22) \\
a_j^k + M(1 - x_{i,j}^k) \geq a_i^k + x_{i,j}^k t_{i,j} + \delta_i & \forall k \in K, \forall i, j \in V & (3.23) \\
\sum_{i \in V^S} x_{i,j}^k - \psi_j \leq \gamma_j^k & \forall j \in T & (3.24) \\
\alpha_k^L - a_j^k \leq M(1 - x_{i,j}^k + \omega_j) & \forall j \in T, i \in V^S & (3.25) \\
a_j^k + \delta_j - \alpha_k^U \leq M(1 - x_{i,j}^k + \omega_j) & \forall j \in T, i \in V^S & (3.26) \\
\sum_{j \in T} \sum_{i \in V^S} x_{i,j}^k \delta_j \leq h^k & \forall k \in K & (3.27) \\
x_{i,j}^k \in \{0, 1\} & \forall i, j \in V & (3.28) \\
y_j, \omega_j, \psi_j \in \{0, 1\} & \forall j \in T & (3.29)
\end{array}
\right.
$$

Table 3.2 presents notation applied to a pricing problem. The pricing problem minimises the cost of the path of a column and its dual price. Constraints are defined to ensure that a route represented by a column is feasible and respects to the problem conditions.

The mathematical formulation of the pricing problem are very similarly with the formulation to define HHC model in Chapter 2. Although, visit assignment constraint has not been used in the pricing problem because the constraint is applied in the master problem. Solving the pricing problem is to find a working path for a worker $k \in K$ where the path should have the cheapest cost. The cost of a path is calculated from travel distances $d_{i,j}$, assigned costs p_j^k, preferences costs ρ_j^k, soft constraint violation costs $\omega_j + \psi_j$, and dual costs $\sigma_j + \pi^k$. The first four costs are the same costs presented in the full model. The additional costs presented here are dual costs which obtain by solving the master problem. σ_j is a dual cost corespondent with the constraint set (3.16) and π^k is a dual cost corespondent with the constraint set (3.15).

The pricing problem include the following constraints. A visit j can be made at most once (3.20). The flow conservation constraints guarantees that once the worker k makes a visit then leaves the visit in order to form a working path (3.21). The worker arrival time a_j must respect the time window (3.22). The arrival time a_j at a succeeding visit j must spare from the arrival time at the predecessor visit i at least the amount of working time δ_i of its predecessor visit i and the travel time between visits i and j. The worker k should make a visit j where it is located in their working region, otherwise the working region soft violation is added to the objective value by having $\psi_j = 1$ (3.24). The worker should make a visit in their time availability, otherwise the time availability soft violation is added to the objective value by having $\omega_j = 1$ (3.25) - (3.26). The total working hours must not exceed their working limits (3.27).

A route generated by pricing problem is a column to be added to the master problem. However, only the column with reduced cost will be selected. The column having reduced cost will have its pricing objective value $R^k < 0$ which

Algorithm 1: Column Generation Algorithm.

```
Data: Problem instance $P = \{K, V\}$
Result: solution
masterProblem = generateMasterProblem(P);
repeat
    masterSolution = cplex.solve(masterProblem);
    dual = getDual(masterSolution);
    for Worker $k \in K$ do
        pricingProblem = generatePricing(k,dual);
        pricingSolution = cplex.solve(pricingProblem);
        if pricingSolution $< 0$ then
            masterProblem.addColumn(pricingSolution);
        end
    end
until no column added;
return finalSolution = cplex.solve(masterProblem);
```

indicates that selecting this column in the master problem will reduce the master problem objective value, which obtain an improved solution.

The literature recommends to solve pricing sub-problem using dynamic programming [53, 123]. Dynamic programming finds solutions of a problem from the previously found solutions to sub-problems [104]. For example, in one worker routing problem, a set of shortest routes to visit n locations can be built the set of shortest routes to visit $n - 1$ locations. From the shortest paths for each set of $n - 1$ locations, the method finds a new location with the smallest overall travelling distance and insert that location to the path. This process usually starts the deduction from small number n, such as $n = 1$, where the optimal solution of the sub-problem can be found easily. This approach has been found to be faster in some previous research than the CPLEX solver, which is the method we chose in our experiment. So, since dynamic programming was not tried here (because branch and price is not the main focus of this work), we suggest that an area of future research is to consider solving the pricing sub-problem using this approach.

Algorithm 1 presents the steps for the column generation algorithm to find the optimal solution. It takes a problem instance $P = \{K, V\}$ where K is a set of workers and V is a set of nodes. The algorithm starts from generating the master problem from the problem instance P. The mathematical formulations used in the master problem are presented in Section 3.1.2. In this initialising step, the master problem does not have any column v_j^k, where no assignment has yet been made.

Nest, the algorithm starts the repeating process. The iterations are made until there is no reduced cost column adding to the master problem. The steps to be made in an iteration start from using the MIP solver to solve the master problem. This step provides column dual reduction cost which will be used to generate the pricing problems. The dual reduction cost is information which is transferred from the master problem to the pricing problem to which visits have not yet been assigned. The algorithm is then built a pricing problem and solve the pricing problem to find reduced cost columns for each worker $k \in K$. The reduced cost columns are added to the master problem if their reduced cost is less than 0, which indicate that the column used in the master problem will reduce the objective value. Then the next iteration is made by repeating to solve the master problem, to generate new pricing problems, to solve pricing problems, and to add reduced cost columns to the master problem. The optimal solution is obtained when there is no reduced cost column can be added to the master problem because adding any other columns will not reduce the objective value.

3.1.3 Computational Result on Column Generation Algorithm

This part presents computational results from implementing a Dantzig-Wolfe decomposition to the HHC problem. Table 3.3 presents the objective value and computational time obtained from using an MIP solver to solve the full problem alongside results obtained with the column generation algorithm. The table

Table 3.3: Objective value and computational time from the MIP solver and the column generation algorithm, applied to 14 HHC problem instances.

Set	Optimal		Column Gen.	
	Obj.	Seconds	Obj.	Seconds
A-01	3.49	7	3.49	134
A-02	2.49	8	2.49	97
A-03	3.00	14	3.00	251
A-04	1.42	5	1.42	85
A-05	2.42	1	2.42	17
A-06	3.55	5	3.55	59
A-07	3.71	1	3.71	12
B-01	1.70	21	1.70	1,31
B-02	1.75	2	1.75	11
B-03	1.72	6,003	1.72	3,580
B-04	2.07	25	2.07	282
B-05	1.82	585	1.82	2,132
B-06	1.62	184	1.62	2,011
B-07	1.79	300	1.79	2,816

presents the results of only 14 instances in order to investigate the algorithm efficiency.

The result shows that the Dantzig-Wolfe decomposition applied within a column generation algorithm is able to match the optimal solution for instance sets A and B. However, the computational time spent on the Dantzig-Wolfe decomposition is much longer than tackling the problem as a whole.

We also applied column generation algorithm to solve instance D-01. The algorithm could solve the master problem in the first iteration but the memory consumption gradually increased when the algorithm entering the loop to generate and solve pricing problems. Therefore, the algorithm was terminated because it reached the computation time limit which was set at 2 hours. The feasible solution given by column generation had very high objective value because the process took only a few iterations before it was terminated, hence the solution did not reach the expected quality.

The literature shows the branch-and-price algorithm was used to solve several mixed integer programming problems [112]. However, the branch-and-price algorithm requires the column generation algorithm as a part of the process, i.e. to solve the problem relaxation. Moreover, our observation shows a large proportion of computational time was used without any satisfying solution can be made. We decided to discontinue the investigation on the branch-and-price algorithm due to our experiment showing the column generation algorithm was a time-consuming process. Instead, we investigated a heuristic decomposition approach which should take less computational time and could perhaps deliver an acceptable solution within the time limit. Next, we discuss heuristic decomposition in the literature.

3.2 Heuristic Decomposition Methods in the Literature

Heuristic decomposition methods aim to find good quality solutions rather than to seek for an optimal solution. As a result, these methods generally find a feasible solution much faster than the traditional decomposition methods. There are implementations of heuristic decomposition methods in the literature.

The heuristic decomposition can be drawn from traditional decomposition approaches such as a cut-and-solve algorithm [49]. The cut-and-solve applies *piercing cuts* which intentionally cut out solutions from the original solution space. Then, the feasible solutions in the removed space are explored in order to find the best solution. The best solution of the removed space then becomes a candidate for the global solution. For the remaining space, the linear relaxation problem is resolved for updating the problem lower bound. The search is terminated when the lower bound from the relaxed problem is greater than the objective value of the known candidate solution. If this is not the case, the process is repeated by adding new piercing cuts, solving the removed space and finding the lower bound of the remainder. The piercing cuts are accumulated in every iteration which results in tightening the problem and reducing the size of the solution space. This approach was found to be robust to solve the asymmetric travelling salesman problem. This technique can solve the travelling salesman problem up to 600 visits. However, this method is potentially not suitable for tackling the HHC problems because this approach starts reducing search space after the integral relaxation of the problem has been solved. However, the experiment in Chapter 2 showed that the MIP solver ran out of memory when tackle the 24 larger instances before starting the integral relaxation process. Thus, when solving the HHC problems, a decomposition algorithm must be applied before start using the MIP solver.

Decomposition can be made into phases which was widely used. For example, an inventory routing problem was decomposed into planning phase for making high level decisions and scheduling phase for making detailed solutions based on a high level plan [32]. Phased decomposition was also applied to a multi-depot location routing problem which considers a problem into location allocation phase and vehicle routing phase [128]. Additionally, a decomposition can be made to partition time-horizon into multiple sub-problems, e.g. a sub-problem of morning shift, etc [14].

The other studies in the literature apply the decomposition method using a form of cluster. An example is to decompose a vehicle routing problem by dividing the geographical region into polar coordinates [119]. This method generates a centre point of the whole problem and splits the problem into polar regions. Locations in a polar region formed a sub-problem. Tours or paths are generated for each sub-problem by using tabu search algorithm to assign available vehicles to visit locations. We choose to elaborate this decomposition approach in subsection 3.2.1.

The heuristic decomposition method can also been used with heuristic algorithm to solve a problem. An example is to use clustering algorithm and local search to solve an overnight security service problem [31]. The service focuses on patrolling of streets and inspection of buildings where a worker must able to response to alert signals. Therefore, a heuristic decomposition tackles the problem in two steps: cluster locations into areas, and build a patrol plan where each route keeps below a required distance from the location cluster. The clustering step is tackled by p-median algorithm. The patrol plan is defined as VRPTW which is solved by an approximation algorithm, AKRed [46]. Another example is to apply an heuristic decomposition method to the saving based ant system [108]. The procedure starts by generating an initial solution using an ant system heuristic, then the initial solution is decomposed to determine sub-problems which are resolved again by an ant system heuristic [57].

We pick the three following examples to explain further as these works have inspired our research.

3.2.1 Decomposition Methods for the Single Depot Vehicle Routing Problems

Taillard [119] proposed two decomposition methods to tackle the single depot vehicle routing problem. The vehicle routing problem (VRP) is to find a solution containing paths for vehicles to start their journey from a central depot, make visits at different locations, and return to their central depot. The assumption of this study is that the depot should be located at the centre of the whole region. A polar decomposition method is proposed to partition into uniform sub-problems. This approach decomposes by partitioning visits into polar regions. The uniform problems are scenarios for which the depot is almost centered and visits are regularly distributed around the depot without forming distinct clusters.

The concept of this decomposition is to generate clusters around the depot while visits are distributed approximately equally between clusters. The polar decomposition can be done in two ways: sector partition and polar region partition. The sector partition is recommended to use with small problem and the polar region partition should apply to the larger problems. Each cluster is then treated as an independent vehicle routing problem and then solved by tabu search algorithm.

This approach shows an example of partitioning a problem into sub-problems and tackle the sub-problems as independent problems. The polar decomposition with tabu search algorithm find high quality solutions for the uniform problems. On the other hand, this decomposition method does not work well when visits are not regularly distributed around the depot or the depot was not located in the centre.

For nonuniform problems, a decomposition method is based on the partition of branches of a minimum weighted spanning tree from the depot to all visits. The assumption of this approach is that visits near to each other should belong to the same branch of the shortest path tree and should belong to the same sub-problem. The method starts from building the minimum weighted spanning tree and then partition the branches so that each sub-problem should have an approximately equal number of visits. This approach is reported that to perform better than polar decomposition.

The two decomposition methods presented in this section are examples of decomposition a problem into sub-problems where each sub-problem is tackled independently. The two decomposition methods are reported to find acceptable solutions with much improvement on computational time. The idea to speed up computational time by decomposition and independent sub-problem solving process will carry on as a selected approach to solve real-world HHC instances. Thus, decomposing a problem this way is the concept of our proposed methods presented in Chapter 4, Chapter 5, and Chapter 6 However, HHC problems are commonly known to be multiple depots problems. Hence, we may require to look for the other methods to decompose the problem as the two methods above are restricted to the single depot problem.

3.2.2 A Cluster-based Optimization Approach for the Multi-depot Heterogeneous Fleet Vehicle Routing Problem with Time Windows

Another way of designing a heuristic decomposition method is to split a problem heuristically. There are several ways to split the problem, such as clustering. Dondo and Cerdá [56] proposed a three-phase heuristic to solve multi-depot heterogeneous fleet vehicle routing problem with time windows (VRPTW). The VRPTW is an extension of the VRP where vehicles to make visits must arrive the location within the appointment time window. In addition, the optimisation solver cannot find the optimal solution of every instance of the multi-depot VRPTW. This three-phase heuristic could be a practical implementation to maximise the use of a mathematical solver to tackle the multi-depot VRPTW.

This heuristic decomposition method by Dondo and Cerdá [56] has three hierarchical phases to generate the routing plan. The three phases are

- Phase I: cluster generation,
- Phase II: cluster assignment and sequencing, and
- Phase III: node sequencing.

Phase I: Cluster Generation

Cluster generation is the key to reducing overall computation time as it clusters locations and defines each cluster $c \in C$ as a node. Each cluster contains several customer locations. Travel distances are defined between a pair of clusters. We can see that the mathematical model of cluster-based problem uses a cluster as a location node, which has smaller size than the problem defined by customer locations.

This phase processes all visit nodes T, where a node $j \in T$ represents a customer to visit, in order to generate clusters which will act as super-nodes in the mathematical model used in phase II. The process in phase I also requires the set of vehicles V, travel distances and time between nodes, service times, and loads of each node. The outcome from phase I is clusters in which their members can form feasible route such that

1. the total loads in a cluster can fit in a single vehicle,
2. there exists a route connecting nodes inside the cluster which satisfy all time window constraints,
3. vehicle waiting time should be minimum, and
4. the average travel duration per node should remain low.

The process follows the following steps:

1. Sort location nodes by increasing the value of the earliest arrival time a_i; if several nodes have the same a_i, sort by increasing value of the latest arrival times b_i. The sorted location list is defined as L.
2. Sort vehicles by decreasing values of the capacity- ratio q_v/cf_v where q_v is a capacity of vehicle v and cf_v is a fixed cost for using vehicle v, and name the sorted one as a vehicle list V.

3. Select the first vehicle v of V, define as a vehicle of a new cluster $c_n \in C$ then remove v from V.
4. Pick a location node j from the list L to the current cluster $c_n \in C$ if j does not make the total load of the cluster exceed capacity q_v of vehicle v and distance of j to the nearest node in the cluster does not exceed the maximum distance allowed d^{max}.
5. If the node j fits into the current cluster, add node j to the cluster.
6. Pick another node from the list L until reaching the end of the list.
7. Repeat step 3-6 to build the next cluster until all location nodes in L are allocated.
8. Calculate the centroid, time and distances between clusters.

Each cluster represents a super node. Therefore, the mathematical formulation of a cluster based problem is built based on clusters. In this case, a super node acts like a single location. Therefore, node components must be addressed, including cluster time window and cluster service time. Cluster time windows are defined by nodes in the cluster such that the earliest start time of the cluster is the minimum earliest start time of all location nodes in the cluster, $\min_{i \in C}(a_i)$, and the latest start time of the cluster is the maximum latest start time of all location nodes in the cluster, $\max_{i \in C}(b_i)$. The cluster service time is the summation of time required by all nodes in the cluster and the travelling time between those nodes.

Phase II: Cluster-based Multi-depot Heterogeneous Fleet VRPTW

This phase assigns clusters to vehicles by using an MIP model. This process takes the set of super-nodes, the set of vehicles, super-node time windows, super-node time and distances as inputs. We can see that the data size is shrink when replacing nodes with super-nodes. Therefore, the optimal solution to the reduced data size should be easier to find, thus, takeing less computation time. The result of this phase is assignments of vehicles to visit nodes (via super-nodes). The solution also allocates the used vehicles to the designated depots, and provides a sequence of clusters on the same tour. The solution in phase II should satisfy all constraints. However, the solution can be improved by adjusting order of visits within a tour which will be processed in phase III.

Phase III: the Single Tour Scheduling Problem

To complete the solution, this process schedules visits inside the tour. This phase assigns vehicle arrival time to visit nodes. This phase tackles every tour provided by phase II where the tour is formulated in the same mathematical model used in phase II and then solved by the MIP solver. The problem can be considered a travelling salesman problem as it requires a vehicle to leave the depot, visit all locations in the tour and return back to the depot.

Result and Discussion

The three-phase method was used to tackle instances in Solomon datasets which considers be homogeneous VRPTW. The method combines visiting nodes into clusters, which results in node reduction in the mathematical problem by 68-90% of the original problems. The outputs from using this approach matched the best known solution. It was shown that Phase II required the most computational time.

The Solomon dataset was modified to be a heterogeneous dataset by changing the capacity of individual vehicles. However, to guarantee that all locations can be visited, the total fleet capacity remained the same. The result from using this approach showed the computational time increases to 2,271 seconds (from 74.15 seconds).

This approach was reported to solve the homogeneous VRPTW successfully [56]. The computational time consumed by this approach is less than the exact method. However, it provided only one sample of the heterogeneous VRPTW which was the main aim of the proposed method.

Generally, this study is an example of using decomposition in multiple depot problems. The decomposition used in this algorithm is not focused on partition a problem, but merges several visits into a single representative location. In addition, this approach also makes a decision that the visits within a cluster must be made by a single vehicle. The MIP solver is used to make decision in phase II and phase III. The process of phase II is to find inter-cluster routes and each route will become an independent MIP problem in phase III. For the HHC problem, using the phase I algorithm to cluster visits might not as simple as the VRPTW because visits in the HHC problem requires workers with satisfactory skills, while any vehicle in the VRPTW can make any visits. Therefore, determining visits to be in the same cluster is almost as hard as solving the whole problem.

There are parts of the solution approach that inspired to our research. The cluster generation process will be adopted in one of our visit partition rules, location based with uniform partition (see page 124). The single tour scheduling (phase III) also an inspiration to the assignment conflict repair process (see Chapter 5). However, there is a slightly difference in assignment conflict repair such that some assignments may be dropped which will become unassigned visits as all assignments cannot be guaranteed to be assigned.

3.2.3 Hybrid Heuristic for Multi-carrier Transportation Plans

We reviews an approach applying a clustering algorithm to build a transportation plan. This study is an example of using an assignment model to solve a part of vehicle routing problem.

A multi-carrier transportation plan is to arrange carriers to distribute goods around the country [78]. The transportation plan should select what carrier to use for a given load where the plan should avoid backward mileage which is the case of a delivery point that is closer to the source than some of the previous delivery points. Loads in this scenario are of three types: Full truck load, Less than a truck load and Groupage. Note that Groupage is smaller size shipments which can be delivered by using either a carrier in a transportation plan or a courier service. The hybrid heuristic method has five steps to produce a transportation plan:

1. Clustering shipments,
2. Create subgroup,
3. Build initial loads,
4. Carrier assignment, and
5. Improve loads.

Clustering Shipment

Shipments are clustered into big regions. The aim is to identify delivery regions across the country where shipments in a cluster are geographically compatible. The delivery region is built by adapted DBScan (Density-Based Spatial Clustering of Applications with Noise) which is one of the automated clustering methods [60]. The basic DBScan generates clusters by using two parameters: the neighbourhood threshold ϵ and the minimum number of points to become a cluster $minPts$. The algorithm starts from an unvisited point p. Next, it seeks for neighbourhood points of p which are points where their distances from p are less than ϵ. If p has neighbourhood points more than $minPts$, it forms a cluster which contains the point p and its neighbourhood points. The point p is also known as the *core* point. This means a selected point p is in a dense area if the number of neighbouring points is more than the $minPts$ threshold. Next, the algorithm selects one of the neighbours of p to find neighbours of the new point. A point might not become a member of any cluster if it is not a neighbour of any core points and its neighbour is less than $minPts$, called noise. Noises are basically non-core points that also are not neighbours of any core points. Noises are not a member of any clusters. Although, the adaptive DBScan may allow noise points to be part of a cluster if they are within ϵ distance to the nearest point in a cluster. Note that some of the noise points may not be absorbed. The adaptation is applied to control the size of clusters. Clusters that have more than 20 location points and are split by applying additional DBScan on them. Too small clusters are avoided by adjusting the minimum point threshold, $minPts$.

Create Subgroups

This part groups locations and considers a group of locations as a single delivery point. In this case, locations within a subgroup should be very close to each other, i.e. 5 mile radius. A subgroup must also have a total load less than or equal to the vehicle capacity. Additionally, the delivery time must be compatible such that all shipments can be served by the same vehicle.

Build Initial Loads

Loads are built by having maximum shipments which are limited by vehicle capacity. The loads built in this part must cover at least all full truck loads and less than truck loads. Groupage may add to the load where there is capacity left from the two load types. The initial loads may violate delivery time which will be fixed later.

Carrier Assignment

At this point, the complex problem is reduced to an assignment problem which assigns loads to carrier companies. The objective function of the model is to find the cheapest transportation cost. The problem is formulated into an integer programming model and it is solved by mathematical solver.

Improve Loads

This stage moves shipments between loads to improve the quality of the plan. This also includes removing delivery time violations and reducing the overall cost, such as by addressing total driving distance and vehicle utilisation. There are four moves that apply in this method: Move shipments between loads in the same cluster, Move shipment between loads in different clusters, Re-sequence subgroup shipment and Re-sequence and adjust shipment in loads. *Move shipment between loads in the same cluster* basically swaps and reassigns shipments in the same cluster iteratively. The move swaps a shipment which has time violation. The swap is repeated until no delivery time violations remain or no further improvements can be made. *Move shipment between loads in different clusters* swaps two shipments between clusters where the distance between shipments is less than 70 miles. Again, this move is applied repeatedly until no delivery time violation remains or no further improvements can be made. *Re-sequence subgroup shipments* is sequencing the deliveries in the subgroup where delivery time violations can be removed. In addition, the re-sequencing process may improve operational cost. *Re-sequence and adjust shipments in loads* explore the shipments in a load and re-sequence those shipments in order to eliminate shipment delivery time violations. Adjustments are made by moving shipments from load mode to parcel mode. The adjustment is made only when the move reducing total operational cost.

Experiment, Result and Discussion

This approach tackled the real-world instances which had number of shipments up to 103. This approach builds loads where the number of loads ranged between 18-25% of total shipments [78]. Loads filled were 60-74% of the vehicle capacity with a small number of delivery time violations (2-6 violations). The delivery time violations were resolved during the improve loads stage which reduced time violation down to 0-1 violation. The total cost of this stage was 0.85-6% higher than the cost from planning in carrier assignment stage but the load capacity was decreasing by about 0.27-6.3%. More importantly, the plan showed improvement compared to the human planner.

From this work, the clustering algorithm with cluster size adjustment mechanism and the use of MIP model as a part of the method are two main features that may help to develop methods to tackle the HHC problem. In Chapter 4, we have found that the geographical region partition made in the HHC instances can be unbalanced which results in some sub-problems are larger than other sub-problems and the larger sub-problems will require significantly longer solving time. Therefore, the mechanism to control the problem size, inspired by this hybrid heuristic, can reduce the overall computation times.

We also have seen the use of assignment model as a part of algorithm to solve the transportation problem where the problem is normally been formulated as a balance flow model. This work has shown that the assignment model can be solved by an open source solver. Later in this thesis, in Chapter 7, we develop an assignment model to solve the HHC problem from which we have learnt from this work that the assignment model is much easier to solve by the MIP solver.

3.3 Summary of Approaches for the Upcoming Heuristic Decomposition Methods

The three selected works from the literature show different ways to tackle real-world problems with heuristic decomposition. These works are our inspiration to develop new heuristic decomposition methods. Chapter 4 presents a decomposition method which partition a problem into sub-problems and each sub-problem is solved by the MIP solver independently, where the solving scheme is borrowed from the decomposition method presented in Section 3.2.1. The scheme has been chosen mainly because solving a sub-problem individually reduces the overall computational times significantly. Adaptations are made to pass assignments and availability information between sub-problems in which assignment conflicts can be avoid because workers in the HHC problems may be used in multiple geographical regions.

The idea of using clustering methods, as presented in examples in Section 3.2.2 and Section 3.2.3, is applied in Chapter 5 which applies different clustering techniques to produce clusters leading to an improved solution. The clustering techniques in Chapter 5 implement mechanisms to control sub-problem size which have been used in the studies presented in previous two sub-sections. The main reason to apply this mechanism is to avoid sub-problems become too large where solving the large sub-problem will take a large amount of computational time. Chapter 6 extends this idea to solve the general WSRP instances where time-dependent activities constraints are enforced.

Finally, Chapter 7 re-formulate the MIP model to solve the HHC problems in the form of assignment model where the idea using assignment model has been used as a part of the method presented in Section 3.2.3. The assignment model is much easier to be solved by the MIP solver than the flow-based model presented in Chapter 2. The re-formulated MIP model in Chapter 7 reduces constraints by redesigning the problem structure, which differs from the approaches in Section 3.2.2.

Chapter 4

Geographical Decomposition with Conflict Avoidance

This chapter proposes a heuristic decomposition algorithm to tackle the home health care problem (HHC). The main objective implementing this heuristic decomposition method is to harness the capability of the mathematical solver when tackle the real-world HHC problems whilst maintaining most of the problem constraints. The problem instances tackled in this chapter are presented in Chapter 2, Section 2.5.

The content of this chapter has been presented in:

- Wasakorn Laesanklang, Dario Landa-Silva and J. Arturo Castillo-Salazar. **Mixed Integer Programming with Decomposition to Solve a Workforce Scheduling and Routing Problem.** In *Proceedings of the 4th International Conference on Operations Research and Enterprise Systems (ICORES 2015)*, pp. 283–293, Scitepress, Lisbon, Portugal, January 2015, Best Student Paper Award.

- Wasakorn Laesanklang, Rodrigo Lankaites Pinheiro, Haneen Algethami and Dario Landa-Silva. **Extended Decomposition for Mixed Integer Programming to Solve a Workforce Scheduling and Routing Problem.** In

Operations Research and Enterprise Systems, Series Communications in Computer and Information Science, Vol. 577, pp. 191–211, Springer, 2015.

Normally, *conflicting assignments* cannot be valid, because they are visits that overlap in the time assigned to the same worker. A solution which contains conflicting assignments, therefore, is strictly infeasible. Solutions which satisfy the mathematical model presented in Chapter 2 do not have conflicting assignments because the route continuity constraint and the assignment time feasibility constraint are enforced. However, the heuristic decomposition proposed here might produce conflicting assignments without a conflict avoidance scheme because each sub-problem is solved independently. Potential conflicting visits can be fixed by assigning them to different workers, delaying some of the conflicting visits so that workers can accommodate them. This kind of technique can be used in online aircraft control system [126]. The method proposed in this chapter will avoid conflicts while building a solution.

4.1 Geographical Decomposition with Conflict Avoidance

Geographical Decomposition with Conflict Avoidance (GDCA), Algorithm 2, is a heuristic decomposition method proposed for tackling the home health care problem. The principle behinds this algorithm is to split a large instance into smaller sub-problems. In this method, a sub-problem is defined by geographical region. Each sub-problem is then solved independently. A sub-problem generated by this method is small enough to be solved by the MIP solver. The algorithm requires conflict avoidance procedure to prevent conflicting assignments. Therefore, sub-problems are sorted and each of sub-problems is solved one at a time. After a sub-problem has been solved, the algorithm updates

Algorithm 2: Geographical Decomposition with Conflict Avoidance (GDCA), Section 4.1.

```
Data: Problem $P = (K, V)$, $K$ is the set of workers and $V$ is the set of nodes
Result: {SolutionPaths} FinalSolution
1  begin
2      Initialisation: for each worker $k \in K$, identify the unavailability period given
       between $\beta_L^k = 0$ and $\beta_U^k = 0$ ;
3      {Sub-problem} $S$ = GeoDecomposition($K$,$V$);                // Section 4.1.1
4      SortSubProblems($S$);
5      for Sub_problem $s \in S$ do
6          sub_sol.addSolution(cplex.solve($s$,$\beta_L$,$\beta_U$));   // Section 4.1.2
7          Update_unavailability ($\beta_L$,$\beta_U$);
8      end
9      FinalSolution = Combine_solutions(sub_sol);                // Section 4.1.3
10 end
```

workforce unavailable period to avoid adding assignments in these periods by the solution of the other sub-problems.

Algorithm 2 shows the steps of GDCA. It takes a full-sized problem instance which consists of a set of workers K and a set of nodes V. First, the initialisation step defines earliest unavailable period β_L^k and latest unavailable period β_U^k. Then, for each worker k problem P is split into sub-problems. Each sub-problem is solved by the MIP solver with conflict avoidance constraints. They are solved in a given sequence to avoid conflicting assignments. The unavailability period, β_L^k and β_U^k, is updated for every sub-problem solved. All sub-problem solutions are combined as a worker cannot have multiple paths. Below, we describe in detail of each main step: geographical decomposition at line 3, conflict avoidance constraints added to the MIP model used by line 6, and combining sub-problem solution step at line 9.

We explain details of GDCA in three parts: geographical decomposition, solving sub-problems with conflict avoidance, and combining solution.

Algorithm 3: Geographical Decomposition (GeoDecomposition), Section 4.1.1.

```
Data: {Problem} $P = \{V, K\}$ where $V = D \cup T \cup D'$ is a set of nodes and $K$ is a
      set of workers.
Result: {Sub_problem} S = $\{S_a\}, \forall a \in A$
begin
    Initialisation: Generate a list of geographical regions $A$ from a set of visit
    nodes $T$;
    for Region $a \in A$ do
        for Visit $j \in T$ do
            if j.Region = $a$ then
                {Visits} $T_a$.add(j);
            end
        end
        for Worker $k \in K$ do
            if k.availableIn(a) then
                {Workers} $K_a$.add(k);
            end
        end
        Sub_problem $S_a$ = generate_problem($T_a$,$K_a$);
    end
end
```

4.1.1 Geographical Decomposition

Geographical decomposition, Algorithm 3, is a process to generate sub-problems. For this case, a full problem is split by geographical regions. The HHC problem has two main components, workers and visits, of which different ways of decomposition are possible.

Visits are partitioned by geographical regions such that visits located in the same region belong to one sub-problem. The geographical data is also guaranteed that visits in the same region are close enough that a worker can make travel between the visits. Workers are decomposed into subsets by their available regions. Thus, if a worker $k \in K$ is available in a region $a \in A$, the worker k is selected as a member of a subset K_a. The same worker k may also be available in another region $b \in A$ where this worker will be a member of a subset K_b as well. From this, we can see that $\bigcap_{a \in A} T_a = \varnothing$, where $\forall T_a \in PT$ but a worker can deliver services to more than one region. Note that PT is a partition of visits and KD is a set of subsets of workers which both have the number of members equal to $|A|$.

Algorithm 3 presents steps to generate a visit partition based on geographical regions. The steps take a set of visit nodes T and extract a list of geographical regions A from the provided visit data. Next, the algorithm searches through each visit $j \in T$ and takes the visit j as a member of subset T_a if the visit j is located in the geographical region $a \in A$. Next, the algorithm finds a subset of worker for region a by search all workers in the set to find who is available to make visits in the region a. If the worker k is available, the algorithm add the worker k to the subset of worker K_a who available in region a. Finally, the sub-problem of geographical region a is generated by the subset of visits T_a and the subset of workers K_a. The algorithm repeats the process for the next geographical region until all regions have been processed.

From the algorithm, we can see that the number of visits between regions are likely to be different because of the nature of visit demands in different geographical regions such as residential area, business centre, etc. The mathematical formulations to define a sub-problem is in the same mathematical formulation used by the full problem (see Chapter 2). Each sub-problem is then solved independently by the MIP solver.

Solving a (sub-)problem using mathematical programming guarantees no intra-problem conflict because assignments in the solution are compiled with the mathematical constraints to be a feasible solution. However, solving several sub-problem independently and using a worker in multiple sub-problems may have high possibility of conflicts between sub-problem solutions or inter-problem conflicts. From this, we require procedure to avoid assignment conflicts by updating worker unavailability from a sub-problem to another. The later assignments can be made to avoid the one that already occupies the worker time.

4.1.2 Conflict Avoidance

The geographical decomposition can generate smaller size sub-problems for the MIP solver. However, solving multiple sub-problems could deliver conflicting assignments. The conflict avoidance arranges sub-problems into an order. Sub-problems are then solved one-by-one and worker availabilities are updated after each sub-problem is solved. Solving the later sub-problem can avoid adding conflicts to the predecessor assignments by adding constraints (4.1) and (4.2). From the condition, there are two availability periods: from the beginning of the time horizon to β^k_L and from β^k_U to the end of the time horizon. Again, the worker k is not available between time β^k_L and β^k_U. Furthermore, only one side of the availability period can be used so that we introduce a binary variable ζ^k where $\zeta^k = 1$ if the duration from the beginning of the time horizon until β^k_L can be used and $\zeta^k = 0$ when the duration starting from β^k_U until the end of horizon is available.

$$a^k_j + \delta_j - \beta^k_L \leq M(2 - x^k_{i,j} - \zeta^k) \qquad \forall k \in K, \forall i \in V^S, \forall j \in V^N \tag{4.1}$$

$$\beta^k_U - a^k_j \leq M(1 - x^k_{i,j} + \zeta^k) \qquad \forall k \in K, \forall i \in V^S, \forall j \in V^N \tag{4.2}$$

Since the sub-problems are solved in order, it is clearly seen that a preceding sub-problem has more worker availability than succeeding sub-problems. Therefore, the solving order affects the quality of the final solution. We set an experiment to find an ordering rule to obtain a good quality solution in Section 4.2 of this chapter.

A solution to solving each sub-problem gives visiting paths i.e. a worker travels from the starting node and visiting nodes to the ending node. Although, a worker might have multiple working paths provided by multiple sub-problems, paths belonging to a worker do not overlap since conflicts are avoided as explained above. However, a worker having multiple paths is not practical because the problem requires a worker to have exactly one working path. Hence, all paths belonged to each worker are combined, which will be explained next.

4.1.3 Combining solutions

Sub-problem solutions are combined in this part of the process, which is presented at line 9 of algorithm 2. Each sub-problem solution provides a visiting path for every worker. However, a worker might have multiple paths because they participate in multiple sub-problems. Multiple paths are then merged into one long path for the whole time horizon.

The combining method is designed based on an assumption that the start location d and the end location d' for a worker k are the same place. This assumption is also applied to all HHC instances.

The process starts from the earliest path Φ_1 and the second earliest one Φ_2. The ending edge of Φ_1 which connects the last visit i and ending node d' and the starting edge of Φ_2 which connects the starting node d to the first visit j are removed, by modifying $x^k_{i,d'} = 0$ and $x^k_{d,j} = 0$. Next, an edge between i and j is selected for worker k by adjusting $x^k_{i,j} = 1$. Thus, Φ_1 and Φ_2 are connected. The process then continues on the connected path and the next earliest path.

The proposed path connection is valid only under the assumption that the start location and the end location for a worker are the same. Additionally, the data instances provided Euclidian distances and the provided distance matrix is symmetric (see Section 2.5). By this assumption, it is clear that $t_{i,j} \leq t_{i,d'} + t_{d,j}$. Hence, the assigned time of visit j remains feasible because

$$a^k_i + t_{i,d'} \leq a^k_{d'} \leq a^k_d < a^k_d + t_{d,j} \leq a^k_j$$

where a_i, a_j, a_d and $a_{d'}$ are the arrival time at visit i and j, start location d and end location d' respectively. This process continues connecting the recently merged path to the next earliest path until a single path for worker k is formed.

Note that it is possible in other WSRP scenarios that the start location and end location for a worker are different (i.e. a worker starts their journey from depot and ends at their home), but we leave this for future work because it is not a current feature of the scenarios.

An alternative approach which might be worth investigation in the future is to have a connection from the last visit of the prior leg to the first visit of the latter one without returning to depot. In this case, it should increase the number of assignments made to a worker. The implementation could be done by defining two choices of sub-problem starting location in sub-problem's model (to be solved at line 6 in Algorithm 2), which are the worker's starting location $d^s \in D$ and the location of the last visit from previous solutions $i \in T$. This also applies to the sub-problem end location so that the two choices are the worker's ending location $d^n \in D$ and the location of the first visit from previous solutions $j \in T$. This adapted model also requires constraints to enforce the consequence of choice, i.e. if the location of last visit j is set as a sub-problem start location, then the sub-problem last visit must be the worker's ending location d^n and duration that visits can be assigned must be latter than the last visit. We did not investigate this alternative approach in this thesis due to the limit of research times and the lower solution quality from the current GDCA implementation. Experiments studying the current GDCA performance will be presented, next.

4.2 Experiments

We conducted an experiment to study the GDCA performance. The flow of the study is depicted in Figure 4.1. The figure outlines the three parts of the experimental design. First, on the left-hand side of the figure, the **permutation study** refers to solving the sub-problems in different orders given by all the different

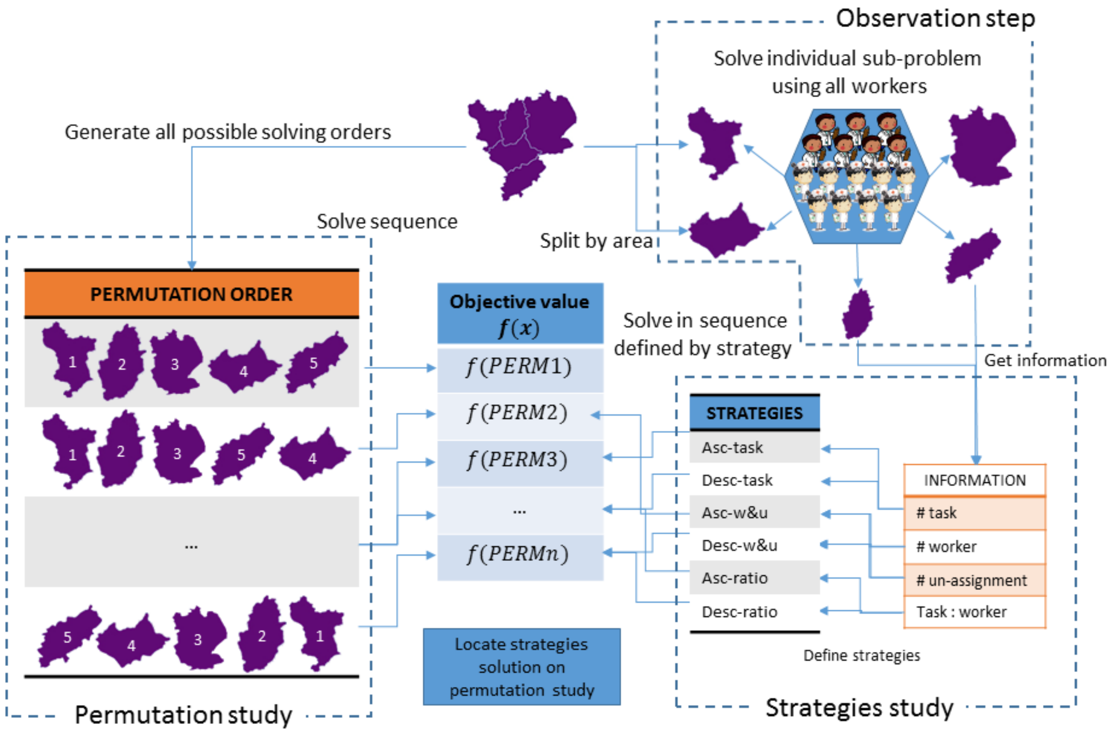


Figure 4.1: Outline of the experimental study in three parts: permutation study, observation step and strategies study.

permutations of the geographical regions. However, trying all permutations is practical only in small problems. Therefore, finding an effective ordering pattern is the second part of the experiment, **observation step** in the figure. This second part solves each sub-problem using all available workforce, i.e. ignoring whether some workers were assigned in previous sub-problems. The third part analysed the results from the observation step in order to define some strategies to tackle the sub-problems. Based on this **strategies study**, some solving strategies were conceived. Listed in the figure are these ordering strategies: Asc-task, Desc-task, Asc-w&u, etc. More details about these ordering strategies are provided when describing the **Observation step** below. Finally, the solutions produced with the different ordering strategies are compared to the solutions produced by the permutation study to evaluate the performance of these ordering strategies.

Permutation Study. Since the number of permutations grows exponentially with the number of geographical regions, we performed the permutation study using only the instances with $|A| = 3$ and $|A| = 4$ geographical regions where the number of permutation is managable. Figure 4.2 shows the relative gap obtained for the small instances that have 3 regions. Each sub-figure shows the results for one instance when solved using the different permutation orders of the 3 regions. Each bar shows the relative gap between the solution by the decomposition method and the overall optimal solution. The figure shows that the quality of the obtained solutions for the different permutations fluctuates considerably. Closer inspection reveals that in these instances the geographical regions are very close to each other and sometimes there is an overlap between them. The result also reveals that some permutations clearly give better results. For example, permutation “1-2-3” for instance A-04, permutations “1-2-3” and “2-1-3” for instance A-05 and permutation “1-3-2” for instance A-07.

Figure 4.3 shows the relative gap obtained for the small instances that have 4 regions. Each sub-figure shows the result for one instance when solved using the permutation orders of the 4 regions. Each bar shows the relative gap between the solution by the decomposition method and the overall optimal solution. Results in Figure 4.3 indicate that some solutions obtained with the decomposition approach using some permutations have a considerable gap in quality compared to the overall optimal solution. The figure also shows that

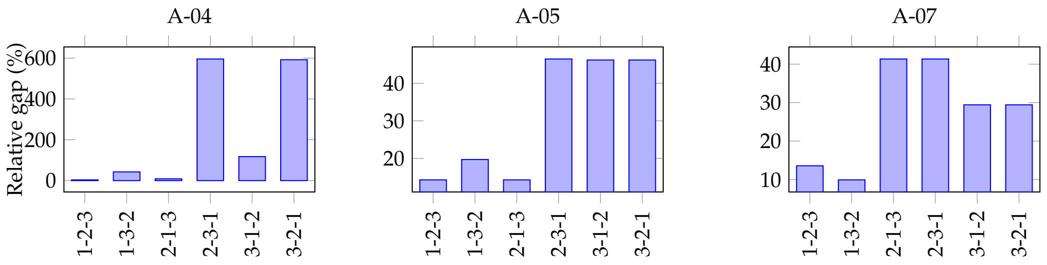


Figure 4.2: Relative gap obtained from solving the 3 instances (A-04, A-05 and A-07) with $|A| = 3$ using the different permutation orders. Each graph shows results for one instance. The bars represent the relative gap between the solution obtained with the decomposition method and the overall optimal solution.

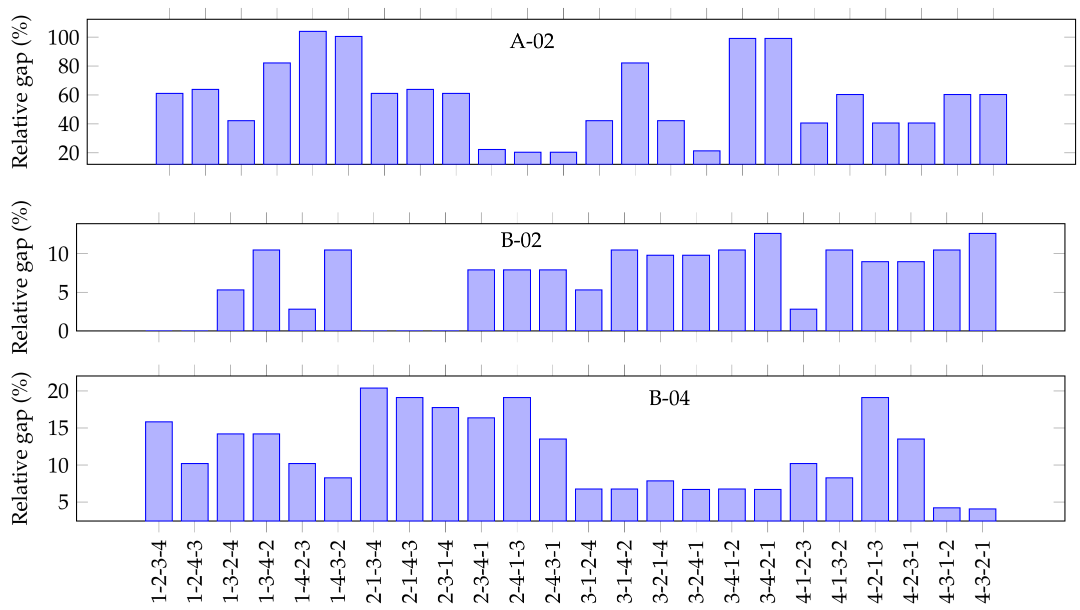


Figure 4.3: Relative gap obtained from solving the 3 instances (A-02, B-02 and B-04) with $|A| = 4$ using the different permutation orders. Each graph shows results for one instance. The bars represent the relative gap between the solution obtained with the decomposition method and the overall optimal solution.

some permutations clearly give better results than others. For example, permutations "2-4-1-3", "2-4-3-1" and "3-2-4-1" for instance A-02, permutations "1-2-3-4", "1-2-4-3", "2-1-3-4", "2-1-4-3" and "2-3-1-4" for instance B-02 and permutations "4-3-1-2" and "4-3-2-1" for instance B-04.

The conclusion from this permutation study is that the order in which the sub-problems are solved matters differently according to the problem instance. More importantly, the results confirm our assumption that some particular permutations could produce a very good result in the decomposition approach. Hence, the next part of the study is to find a good solving order.

Observation step. Here we solve each of the sub-problems using all available workers and collect the following values from the obtained solutions: number of visits in the sub-problem (# visit), minimum number of workers required in the solution (# min worker), number of unassigned visits in the solution (# unassigned visit) and the ratio of visits to worker in the solution (visit/worker ratio). Then, we defined six ordering strategies as follows. Increasing number

Table 4.1: GDCA solution gap to the optimal solution of 14 smaller instances by six ordering strategies.

Instance	Asc-Task	Desc-Task	Asc-w&u	Desc-w&u	Asc-Ratio	Desc-Ratio
A-01	**24.05**	62.04	**24.05**	62.04	**24.05**	62.04
A-02	**41.87**	81.94	107.40	45.07	46.79	48.60
A-03	**151.59**	255.46	**151.59**	255.46	204.73	154.05
A-04	**8.66**	117.57	**8.66**	117.57	**8.66**	117.57
A-05	**14.28**	46.20	**14.28**	46.20	**14.28**	46.20
A-06	**0.00**	5.48	**0.00**	5.48	**0.00**	5.48
A-07	41.34	29.43	**13.57**	29.43	41.34	29.43
B-01	17.51	**5.07**	14.07	14.66	17.91	5.15
B-02	10.46	7.89	10.46	**0.00**	5.30	8.95
B-03	30.70	**19.77**	85.78	56.55	83.99	21.26
B-04	10.20	**6.70**	10.20	**6.70**	19.11	6.76
B-05	207.22	160.06	**126.53**	271.78	158.14	130.43
B-06	61.78	55.28	54.99	**36.46**	151.54	114.09
B-07	140.24	126.60	**104.48**	126.86	244.32	182.41
Average	54.28	69.96	**51.86**	76.73	72.87	66.60

Bold text refers to the best solution.

of visits in the sub-problem (Asc-task); decreasing number of visits in the sub-problem (Desc-task); increasing sum of minimum workers required and unassigned visits (Asc-w&u); decreasing sum of minimum workers required and unassigned visits (Desc-w&u); increasing ratio of visits to worker (Asc-ratio) and decreasing ratio of visits to worker (Desc-ratio).

Strategies study. The GDCA approach is again executed using the 6 ordering strategies listed above to tackle the sub-problems in each problem instance. The results are presented in Table 4.1 which shows the relative gap for the 14 small instances in the A and B groups. Note that each value represents the relative gap obtained with each strategy.

Table 4.1 presents GDCA solution relative gap to the optimal solution of the 14 smaller instances when applying six different ordering strategies. These 14 smaller instances are the HHC instance sets A and B where the optimal solution can be found by solving the problem as a whole. The results in the table are grouped by instance sets and the last row presents average relative gaps to the optimal solution. Overall, the decomposition technique with ordering strategies gives solutions with relative gaps up to 270% with 65% on average. The results show that some of the ordering strategies are more likely to produce

Table 4.2: Relative gap (%) of best permutation VS. best strategy.

Set	B.Perm	B.Strt	Set	B.Perm	B.Strt
A-04	3.53	8.66	A-02	20.41	41.87
A-05	14.28	14.28	B-02	0	0
A-07	9.91	13.57	B-04	4.07	6.70

better solutions than others. The best performing ordering strategy is Asc-w&u that gives 8 best solutions considering all 14 small instances. The average gap for the ordering strategies Asc-task, Desc-task, Asc-w&u, Desc-w&u, Asc-ratio and Desc-ratio are 54.28%, 69.96%, 51.86%, 76.73%, 72.87% and 66.60% respectively. On this occasion, we considered solutions with gap more than 100% as poor solutions. Thus, the GDCA with strategies did not perform well to solve instance A-03, B-05, and B-07 as six strategies cannot find a solution with less than 100% gap to the optimal solution. Table 4.2 shows a comparison of relative gap between the best permutation order (see **Permutation study**) and the best ordering strategy. Only six instances have been used in this comparison because each of six instances has less than 5 sub-problems where all the permutation can be made. There are differences between the best strategies and the best permutation at maximum of 21.46%. Two out of six solutions (instance A-05 and B-02) of the best ordering strategy match the solution from the best permutation. This shows that the ordering strategies are able to work well in other problem instances.

The decomposition method is also able to find solutions for the large instances. The results from using the decomposition technique with the 6 ordering strategies on the large instances are presented in Table 4.3. The table shows the objective values of the obtained solutions as relative gaps cannot be computed because the optimal solutions are not known. The values in **bold** are the lowest cost (best objective value) obtained among the six strategies. The table shows that as a whole, Desc-task gives six best solutions, Desc-ratio gives four best solutions, Asc-w&u gives two best solutions, Desc-w&u and Asc-task give

Table 4.3: Objective value obtained from solving large instances using six ordering strategies.

Instance	Asc-task	Desc-task	Asc-w&u	Desc-w&u	Asc-ratio	Desc-ratio
D-01	1,688	496.45	1,549	765.48	1,301	**240.98**
D-02	860.50	**372.94**	496.47	495.44	984.98	732.97
D-03	2,625	3,213	2,619	3,837	**1,691**	3,839
D-04	312.43	418.89	303.45	**283.91**	314.42	420.41
D-05	408.42	243.89	1,113	253.91	401.45	**241.89**
D-06	**307.55**	1,411	946.60	1,583	634.05	1,729
D-07	1,113	753.28	**292.55**	604.01	293.53	1,077
F-01	73,287	64,305	71,430	72,040	75,761	**63,681**
F-02	81,853	**73,291**	76,460	80,570	86,906	74,860
F-03	141,060	**115,235**	140,258	120,715	148,092	116,011
F-04	111,671	102,994	105,262	109,411	113,557	**91,670**
F-05	127,476	**101,438**	113,403	105,284	112,995	103,156
F-06	105,595	**76,007**	88,702	84,050	107,281	84,050
F-07	199,160	**176,541**	194,525	178,387	218,059	178,387
Average	30,266	**25,599**	28,478	27,083	31,011	25,719

Bold text refers to the best solution.

one best solution while the Asc-ratio gives no best solution. On average, the Desc-task strategy gives the lowest cost solution, around 17.45% less than the highest average cost strategy (Asc-ratio).

Finally, we use statistical test to validate our choice from the observation on the number of the best solution and the lowest average solution that Desc-task is the best ordering strategies for GDCA. Thus, Friedman ANOVA has been applied to measure the differences in objective values of between the six ordering strategies. Table 4.4 presents result from Friedman ANOVA which is in the form of in two sub-tables. The first sub-table shows the statistic value that the calculated statistic value $\chi^2 = 11.335$, the degree of freedom is 5, and the p-value is .045. With significant level $\alpha = .05$, the test shows that the mean ranks between six ordering strategies are different significantly. The second sub-table presents the mean ranks of the six ordering strategies where the lower rank indicates the better solution. The mean rank confirms that Desc-task is the best ordering strategies amongst the proposed six methods as it has the lowest mean rank at 2.89. The highest mean rank ordering strategies is Asc-ratio where the value is 4.16. Therefore, in term of solution quality, we select Desc-task in the GDCA to compare with the other algorithms in Chapter 5 and Chapter 7 (full

Table 4.4: Friedman statistical test and mean ranks of objective value on six ordering strategies of GDCA. The lower mean rank presents better solution quality.

Friedman Test

N	28
χ^2	11.335
df	5
p	.045

Mean Ranks

Feature	Ordering	
	Asc	Desc
task	4.11	2.89
w&u	3.05	3.41
ratio	4.16	3.38

comparison presented in Chapter 7).

Figure 4.4 shows, according to the problem size, the computation times used by the decomposition approach using the different ordering strategies and the time used to find the overall optimal solution. Each sub-figure presents the problem instances classified by their size (number of items is $|T| + |K|$). Each line represents the time used by the ordering strategy in solving the group of 14 problem instances. As noted before, the time to find the optimal solution represented by $\boxed{- - -}$ is available only for the small instances. For the instances which are smaller than instance B-06 (89 items), the computation time used by the decomposition method is not much different from the time used to find the optimal solution. The computation time used to find the optimal solution grows significantly for instances B-06 to B-03. The reason behind this is an increase in the problem size where the instances A-05 to B-04 have between 32 and 64 items while the four instances B-06, B-07, B-05, and B-03; have from 89 items to 103 items. Note that for instance B-03 which has 109 items, the MIP solver uses 5,419 seconds for finding the optimal solution. For the latter four instances, GDCA used less computational times than a half computational times of the MIP solver.

For the large instances, it is shown that the computation time used by the decomposition method starts from 17 minutes (1,060 seconds) to above 6 hours

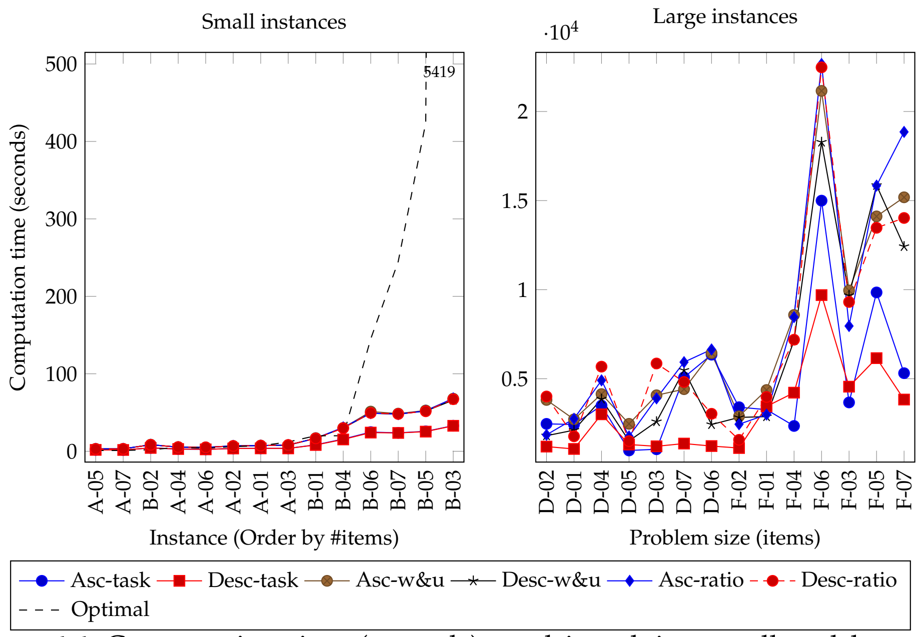


Figure 4.4: Computation time (seconds) used in solving small and large instances. Each sub-figure corresponds to a problem size category (small and large). Instances are ordered by The problem size (#items) which is the summation of #workers and #visits. Each graph presents the computation time used by the decomposition method with the different ordering strategies (line with markers) and the time used for producing the overall optimal solution (dashed line) when possible.

(22,478 seconds). Also, for the large instances the average computation time used by six strategies are 4,620 seconds; 3,098 seconds; 7,451 seconds; 6,348 seconds; 7,640 seconds; and 7,048 seconds respectively. The result shows the average processing time of Asc-task and Desc-task are significantly less computation time than the other strategies. This is because these ordering strategies do not require an additional process to retrieve information about the problem.

Again, we use Friedman statistical test to validate our computational time observation. Table 4.5 presents the result of the statistics in two sub-table. The first sub-table shows the statistic value of testing six strategies on 28 instances: A, B, D, and F. The calculated value $\chi^2 = 74.484$, degree of freedom is 5, and the p-value is less than .01. The Friedman test draws a conclusion that computational times between six ordering strategies are significantly different at significant level $\alpha = .05$. The second sub-table presents mean ranks of six or-

Table 4.5: Friedman statistical test and mean ranks of computational time on six ordering strategies of GDCA. The lower mean rank presents better solution quality.

Friedman Test

N	28
χ^2	74.484
df	5
p	$<.01$

Mean Ranks

Feature	Ordering	
	Asc	Desc
task	2.21	1.46
w&u	4.91	4.07
ratio	4.48	3.86

dering strategies where the lower mean rank refers to the less computational time used. The result confirms that Desc-task is the fastest strategies with its mean rank at 1.46 and the second fastest strategies is Asc-task with the mean rank at 2.21. The other four strategies have very similar computational time where their mean ranks are between 3.86 to 4.91.

Hence, considering both solution quality and computation time, it can be concluded that Desc-task should be selected for large instances because it finds solutions which are overall the best in quality, provided by the objective value mean rank, and also which is the fastest ordering strategy, as shown in the computational time mean rank.

4.3 Geographical Decomposition with Neighbour Workforce

One aspect of GDCA that can be improved is allowing workers to make visits outside their working regions. Applying this will reduce the overall objective value because of the reduction in the number of unassigned visits. Making visits outside the working region was prevented during geographical decomposition because a sub-problem must have only workers who are available in sub-problem region. We list the number of visits and the number of workers grouped by regions in Appendix B.

Originally, the full problem defines working region as a soft constraint so that assigning a worker to make visits outside its regions is allowed by having additional cost. Thus, this practice is valid to accommodate more visits.

Therefore, we intend to reduce the number of unassigned visits by allocating visits to workers who are not available in the region. For this, sub-problems should have additional workers which are recruited from neighbouring regions. Ideally, using all workers in all regions should give the best possible outcome. Unfortunately, the MIP solver cannot handle a problem with such a large number of workers. Thus, number of workers from neighbour regions is added in order to match the total number of visits in a sub-problem.

A neighbour worker is defined by a neighbour score $N(k, P)$ which is calculated by the number of visits the worker k can make and the distance from the worker departure location to the centre of the region. The function is presented in (4.3).

$$N(k, p) = d_{k,c(p)} + \sum_{j \in T} (1 - \eta_j^k) \tag{4.3}$$

where $c(p)$ is a location in the centre of sub-problem p and η_j^k is a binary qualification parameter of worker k to visit j ($\eta_j^k = 1$ if worker k can make visit j, $\eta_j^k = 0$ otherwise). This scoring is only applied to workers who are not available in the selected region, i.e. neighbour workers for sub-problem P. The workers with the lowest score $N(k, p)$ are added to the sub-problem until the total number of workers is equal to the total number of visits in the sub-problems.

Neighbour workers are added to sub-problems where number of workers is less than number of visits. We summarise steps to find additional workers below.

For each sub-problem,

1. Determine number of additional workers to add to a sub-problem p by

$n = |T_p| - |K_p|$ where T_p is a set of visits and K_p is a set of workers of the sub-problem. If $n > 0$ then do all following steps 2 - 5, otherwise does not require additional worker and begin sub-problem solving (step 5).

2. Calculate neighbour score of workers who are not available in p, denoted the set of these workers as $K_p^{'}$, using the function (4.3).

3. Sort workers in $K_p^{'}$ by their neighbour score from low to high value.

4. Add n lowest score workers to the worker set K_p.

5. Start solving the sub-problem and update worker's unavailable period.

After adding workers to a sub-problem, the method solves the sub-problem with conflict avoidance constraint and updates worker's unavailable periods. Then the method tackles the next sub-problems in the ordering list.

The instances that require a neighbour workforce are instance sets D and F as presented in Table 4.6. For each instance, the table shows in columns two and seven, the number of regions that required additional workers. Columns three and eight give the average ratio between the number of available workers and the number of locations. Columns four and nine show the improvement obtained in the objective function value when using this process of adding neighbour workforce. The result shows that additional neighbour workforce is more beneficial to the set F instances for which the cost decreased by up to 75.63% from the solution without additional neighbour workforce. On average, the solution cost decreases by 39.55%.

On the other hand, some of the set D instances did not benefit from the additional workforce, which is an indication that such instances have the right number of workers for the demand. This experimental result suggests that in the set of F instances, the workforce might not be distributed well across regions according to the demanded visits, which then causes problems when

Table 4.6: Objective value improvement and average ratios between number of visits and number of workers for instance sets D and F. The second column shows the number of regions having not enough workers. The third column shows average workforce/locations ratio in regions that workers is less than visits. The forth column shows average decrease of the on objective function after having additional workers

Instance	$\lvert A\rvert$	$\lvert M\rvert$	Ratio	Decrease	Instance	$\lvert A\rvert$	$\lvert M\rvert$	Ratio	Decrease
D-01	12	5	73.76%	41.17%	F-01	44	13	39.51%	70.57%
D-02	11	4	75.04%	50.78%	F-02	45	18	40.92%	66.79%
D-03	14	5	67.13%	-5.87%	F-03	53	22	31.87%	67.33%
D-04	14	5	76.77%	0%	F-04	46	17	40.09%	70.04%
D-05	14	4	73.08%	0%	F-05	58	19	37.97%	56.29%
D-06	14	5	64.72%	0.10%	F-06	43	13	44.58%	75.63%
D-07	14	7	69.31%	7.34%	F-07	63	23	33.73%	53.57%

#Regions is number of regions that workers is less than visits.
Ratio is average of proportion between workers and locations.
Decrease is average of decreasing on objective function calculated by $\frac{(originalObj-addedWorkerObj)}{originalObj}$
$|A|$ is a number of all regions, $|M|$ is a number of regions that the number of workers is less than the number of visits.

decomposing the problem by regions.

4.4 Conclusion

To summarise, this chapter presents a decomposition method to solve the home health care problem. *Problem decomposition* is made by geographical region. The approach avoids having conflicting assignments by solving sub-problems in sequences. Each sub-problem solution gives only a part of a working path. A full working path is then built from multiple parts during the *combining sub-problem solutions* step. Finally, adding neighbour workforce is applied as an extension of this method. The idea is to add other workers from neighbour regions to take unassigned visits.

There are three main studies presented in this chapter: permutation study, strategies study, and neighbour workforce study. The permutation study aims to find the best outcome of applying GDCA method as it searches on every possible sub-problem permutation order. The strategies study compares ordering rules and finds the best ordering rule for general uses. Finally, neighbour workforce is used to improve the solution quality.

The permutation study shows that GDCA is able to find optimal solution. However, the condition depends on a defined geographical region. Furthermore, the sub-problem sequence used indeed affects the solution quality as shown in A-04 where the solution gap ranged from 3.41% to 80%. This is the main reason to select sequences which provide a low objective function.

The strategies study finds effective ordering rules which could give a higher quality solution. The study is crucial as using permutation is very limited due to the number of permutations growing exponentially. Thus, the study tests six ordering strategies. The result suggests ordering the sub-problem by the number of visits gives the lowest average objective function and consumes less computational time. Furthermore, the result is compared back to the permutation study to find differences to the best possible outcome of decomposition method. It shows strategies could match the best permutation on two instances while the rest has slight differences up to 4% of relative gap.

Neighbour workforce is an extension to the GDCA which focuses on improving solution quality. The test only applies to instance sets D and F as the number of workers in the sub-problems in these instances is less than the number of required visits. The study shows the extension is able to reduce objective function down to 75% of the original value.

From these studies, we have seen that this approach is able to find a feasible solution especially on instance sets D and F where solving them as a whole problem is impossible. However, we have seen high objective value on several test instances. The reasons behind this could be due to the approach of avoiding conflicts. Thus, in the next chapter, we introduce a potential alternative for dealing with conflicts.

Chapter 5

Decomposition with Conflict Repair

In the previous chapter, geographical decomposition with conflict avoidance (GDCA) was shown to have potential for solving the larger problem instances. However, we can see that the solution quality depends on having the right sub-problem solving sequence. In fact, because it was not found that a particular sequence dominated the others, this indicates that finding the right sequence would not be be practical. Therefore, we propose a sequence free decomposition technique that not only takes less parameters by removing solving sequences but also does not require conflict avoidance constraints (4.1) - (4.2), which is not required by the main problem definition. Later in this chapter, we propose a geographical decomposition with conflict repair (GDCR) and then present an improved version of GDCA, a repeated decomposition with conflict repair (RDCR). These algorithms aim to solve the home healthcare problem presented in Section 2.5.

The content of this chapter is to be appear in:

- Wasakorn Laesanklang and Dario Landa-Silva. **Decomposition Techniques with Mixed Integer Programming and Heuristics to Solve Home Healthcare Planning Problems.** *Annals of Operations Research*, Online First, 2016..

5.1 Repairing Process in the Literature

The term "Repairing" meaning to correct infeasible solutions has been used mostly in the context of evolutionary algorithms [8]. A repairing method in genetic algorithms recombines an infeasible solution to generate a feasible one [105]. For a scheduling problem, a systematic repair approach was proposed using a bias heuristic to tackle schedules with excessive work-in-progress [130]. An iterative heuristic repairing method had been proposed in the scheduling problem as part of an automated scheduling and rescheduling system [131]. Basically, the method relaxes some constraints when constructive methods found difficulty in completing a feasible solution. The iterative repairing processes in this method is applied iteratively until the solution quality is satisfactory. Another repairing technique was implemented to support the local search algorithm for tackling the job-shop scheduling problem [98]. The use of this repairing technique allows local search moves to continue its search when the move finds an infeasible solution.

Applying a repairing process is not commonly known for mathematical programming based decomposition methods because most approaches, i.e. Benders' decomposition, only generates a solution from the feasible region. Therefore, solutions obtained by Benders' decomposition method do not need to be repaired. However, the solution solved by the decomposition approaches proposed in this chapter may result in an infeasible solution when using the full model. The proposed decomposition approaches use the MIP solver to solve every sub-problem which is generated from decomposing the full problem. Thus, a solution to the sub-problem is feasible only for the sub-problem, but when combining all sub-problem solutions, the combined solution becomes infeasible as sub-problems are solved independently. As a result, we use conflicting assignments repair to fix solutions provided by the decomposition stage.

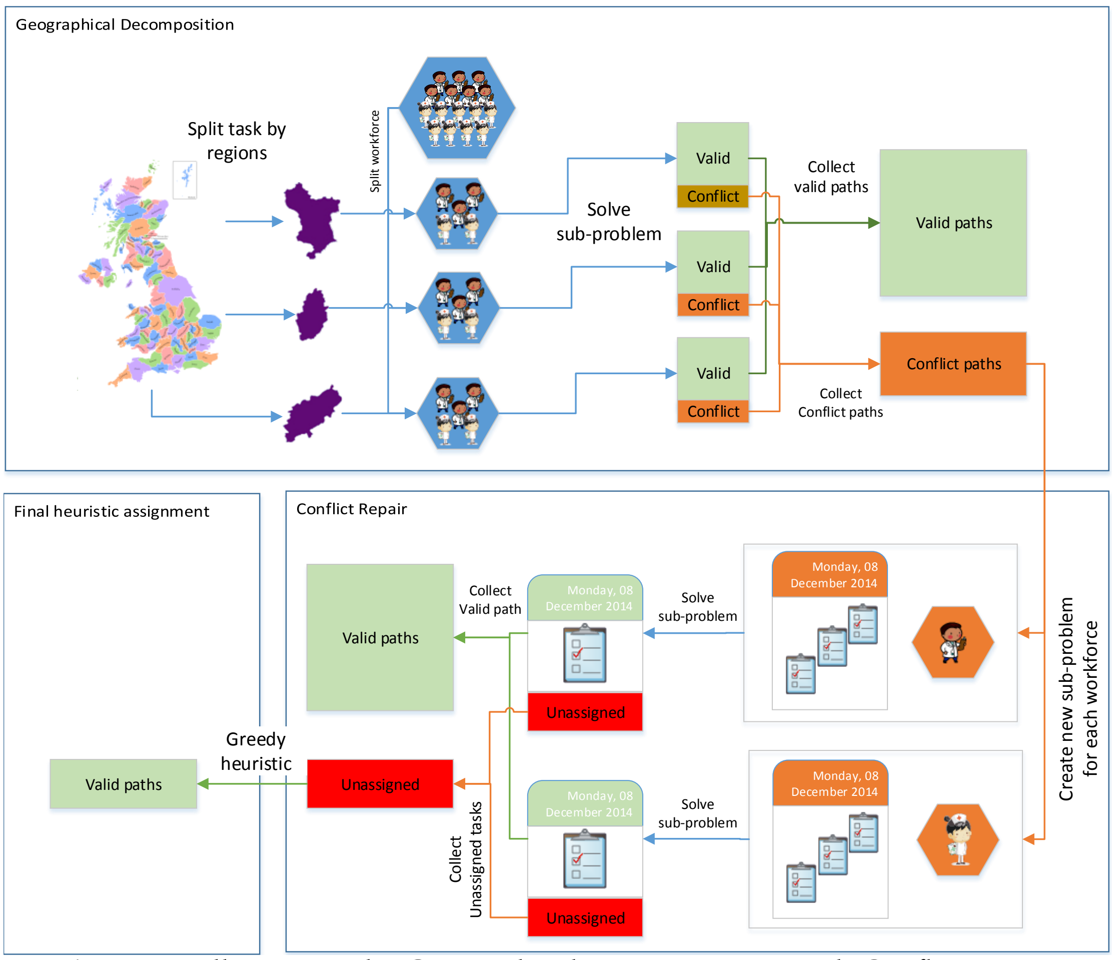


Figure 5.1: Illustrating the Geographical Decomposition with Conflict Repair Approach.

5.2 Geographical Decomposition with Conflict Repair

This section describes a geographical decomposition with conflict repair (GDCR) approach which consists of three stages: geographical-based decomposition, conflicting assignments repair and heuristic assignment. The first two stages complete most of the visits assignments in the problem instance, but the final heuristic assignment is crucial to complete the whole solution.

Figure 5.1 shows the outline of the proposed Geographical Decomposition with Conflict Repair (GDCR) approach. The upper rectangle in the figure illustrates the geographical decomposition, the lower right rectangle illustrates

Algorithm 4: Geographical Decomposition and Conflict Repair

```
Data: Problem $P = (K, V)$, $K$ is the set of workers and $V = D \cup T \cup D'$ is
      the set of nodes
Result: {SolutionPaths} FinalSolution
1  begin
       /* Geographical Decomposition                                    */
2      {Problem} $S$ = ProblemDecomposition($K$, $V$) // Algorithm 5
3      for $s \in S$ do
4          sub_sol($s$) = cplex.solve($s$)
5      end
       /* Conflicting Assignments Repair                                */
6      {Problem} $Q$ = ConflictDetection(sub_sol, $S$)
7      FinalSolution.add(NonConflict(sub_sol))
8      for $q \in Q$ do
9          cRepair_sol($q$) = cplex.solve($q$)
10     end
11     FinalSolution.add(cRepair_sol)
12     {UnassignedVisits} $T'$ = $T$.notAssignedIn(FinalSolution)
13     Update_AvailableWorkforce($K$)
       /* Heuristic Assignment                                          */
14     {Assignment} $HS$ = HeuristicAssignment($T'$,$K$) // Algorithm 7
15     FinalSolution.addAssignment($HS$)
16 end
```

the conflicting assignments repair and the lower left rectangle illustrates the final heuristic assignment. Each part summarises the outline of the process to retrieve a final solution.

Algorithm 4 outlines the GDCR method which takes a problem instance and generates a solution by assigning paths to the workforce. The algorithm shows the three stages executed in sequence: geographical decomposition (lines 2-4), conflicting assignments repair (lines 6-9) and heuristic assignment (line 14). We now proceed to describe these three processes in subsections 5.2.1, 5.2.2 and 5.2.3 respectively. Each sub-problem is defined by the MIP model presented in Chapter 2 and solved to optimality by the MIP solver.

The GDCR takes the idea of decompose a problem by geographical region from GDCA. The changes made from GDCA is to remove the use of sub-problem ordering strategies and the workforce unavailable time constraint from the model.

Algorithm 5: Problem Decomposition

```
Data: {Workers} $K$, {Nodes} $V = D \cup T \cup D'$
Result: {Problem} $S$ is a collection of sub-problems.
1 begin
2     {{Visits}} TP = VisitPartition($T$);
3     for $T_n \in TP$ do
4         {Workers} $ws$ = WorkfoceSelection($K$,$T_n$);
5         $S$.add(subproblem_builder($T_n, ws, D, D'$));
6     end
7 end
```

Thus, after sub-problems are solved, the solution contains conflicting assignments which are the case when a worker is assigned to two visits in overlapping time. The paths containing conflicting assignments are then marked as conflicting paths. These conflicting paths are then repaired by the conflict repair process to get paths that satisfy all constraints to the full problem. The conflict repair chooses some of conflicting assignments to become unassigned visits. Finally, heuristic assignment tackles these unassigned visits by assign these visits to the most efficient available workers. Then, the algorithm returns the solution of the HHC problem.

5.2.1 Problem Decomposition

As illustrated in figure 5.1, the problem decomposition stage in GDCR decomposes the problem into several smaller sub-problems separated by geographical region. This is done exactly as in GDCA, i.e. the sub-problems are defined by the geographical regions. The main goal of problem decomposition is the sub-problem building process. The process involves two main components: visits and workforce. Algorithm 5 outlines this stage. The set of visits are partitioned (line 2) and workforce is selected for the visits in each partition T_n (line 4). The sub-problems are generated by *subproblem_builder* which basically collects related data for the sub-problem.

Visit Partition

Visits are mainly partitioned by geographical regions and then partitioned visits in high demand geographical regions into multiple sub-problems where the number of visits in sub-problems are almost equal. In this thesis, sub-problems with approximately equal number of visits are called *uniform partition*. Algorithm 6 presents a visit partition by geographical decomposition. It takes the set of visiting nodes T and returns a partition set TP with no partition element larger than *subProblemSize*. Given A is a set of geographical regions of a problem instance. The algorithm starts by grouping visits by regions, defined as T_a where $a \in A$ (line 3 - 7). Next, the procedure partition each of visit group T_a using uniform partition if the T_a is larger than the provided *subProblemSize* (line 8-15) . Finally, groups of visits where their members are less than the sub-problem size are added to returning list *TP*. Our basic assumption is that visits located in the same region should be grouped together. Thus, all visits located in each region $a \in A$ are added to the subset T_a. Note that some regions such as high-density residential areas may contain many more visiting nodes than the *subProblemSize* which will become large subsets. The algorithm splits a large subset (assume here as T_a) into smaller subsets by distributing visits approximately equal number until the number of locations of the new subsets W is less than *subProblemSize*. The second partition level is a tool to control the size of sub-problem so they can be solved to optimality by the MIP solver.

Workforce Selection

The set of workers cannot be partitioned because a worker may be associated with multiple geographical regions. A worker can be deployed in every sub-problem involving his selected geographical region. This part works exactly as in GDCA. Each sub-problem is defined by the same MIP model presented in Chapter 2. Therefore, a worker may be assigned to every sub-problem they can

Algorithm 6: Visit Partition: Geographical Decomposition

```
Data: {Visits} $T$, subProblemSize
Result: {{Visits}} TP = $\{T_n | n = 1, \ldots, |S|\}$; Visits partition set
begin
    {Region} $A$ = readRegion($T$);
    for $j \in T, a \in A$ do
        if $j.location.in(a)$ then
            $T_a$.add($j$);
        end
    end
    for $a \in A$ do
        if $|T_a| > subProblemSize$ then
            {Visits} $W$ = uniformPartition($T_a$,subProblemSize);
            TP.addAll($W$);
        else
            TP.add($T_a$);
        end
    end
end
```

participate and each sub-problem $s_i \in S$ is solved independently which results in a worker being assigned to different visits at the same time in different sub-problems. Conflicting assignments will not be allowed in the final solution, therefore, the conflicting assignments repair must resolve these assignments which is explain next.

5.2.2 Conflicting Assignments Repair

This process takes the solution from solving each of the $s_i \in S$ sub-problems and identifies conflicting assignments to form conflict sub-problems. For each path Φ_i in the solution of sub-problem s_i, the algorithm searches for all the conflicting paths in the other sub-problem solutions. A conflicting path is any other path Φ_j that uses a worker also used in Φ_i. Then, if conflicting paths exist for a worker, they are removed from the sub-problem solutions and put together in a conflict sub-problem. Each conflict sub-problem is defined by the MIP model presented in Chapter 2 and corresponds to exactly one worker and all visits that were in the set of conflicting paths. Each conflicting sub-problem is then solved with the MIP solver as shown in line 9 of Algorithm 4. Solving a conflicting sub-problem gives a single valid path for the worker but perhaps with some unassigned visits due to the optimisation process. The heuristic assignment process described next seeks to incorporate these unassigned visits into the overall solution.

There are also alternative approaches which could be used to repair the conflict assignments which were not implemented in this thesis. The alternative approaches include enumeration approach, dynamic programming, branch and bound, etc. These approaches may reduce the time to repair conflicting assignments. However, these approaches have not been implemented in this thesis because most of the computational time was spent to solve the sub-problems from the problem decomposition stage and conflict assignment repair usually needed only 5% of overall computational time (see Figure 5.4 in Section 5.3). Therefore, our focus has been on the computational time reduction in the decomposition steps.

5.2.3 Heuristic Assignment

At this stage, heuristic assignment tackles the set of unassigned visits T' from the conflicting assignments repair. Algorithm 7 outlines this simple greedy approach. The algorithm finds the most cost efficient worker $k \in K$ to make visits $j \in T'$. The assigning condition is that the new assignment cannot conflict with any visit that has been allocated. However, assigning a worker to the visits located outside their available geographical region is allowed with soft constraint penalty cost as explained in Chapter 2.

Algorithm 7: Heuristic Assignment

```
Data: {UnassignedVisits} $T'$, {Workers} $K$
Result: {Assignments} $HS$
1 begin
2     for Visit $j \in T'$ do
3         Worker $k$ = bestCostForVisit($K$,$j$);
4         $HS$.addAssignment($j$,$k$,startTime($j$));
5     end
6 end
```

5.3 Experimental Study on the Stages of Geographical Decomposition with Conflict Repair

We mainly investigated how the three stages in the GDCR method contribute to generating the final solution to the whole problem instance. For this, we scrutinized the proportion of assigned visits, travelling distance, monetary cost and computation time for each of these stages when producing a solution.

Figure 5.2 shows the proportion of visits assigned in each of the three stages. Each of the 42 stacked bars corresponds to 100% of the number of visits in the corresponding problem instance. Each stacked bar has four parts: decomposition, conflicting assignments repair, heuristic assignment and unassigned visits. Each part indicates the proportion of visits assigned in the corresponding stage of GDCR. On average, the proportion of visits assigned by decomposition, conflicting assignments repair and heuristic assignment were 26.34%, 47.50% and 25.82% respectively. Only 0.34% of the visits were left unassigned. In general, the conflicting assignments repair stage achieves the largest proportion of successful assignments, except for instance set C. This set has instances in which most visits have a long duration of 6-9 hours. Therefore, it is likely that workers could take only one visit in the solution to each sub-problem. Therefore, the conflicting assignments repair stage was less successful in solving conflicting sub-problems.

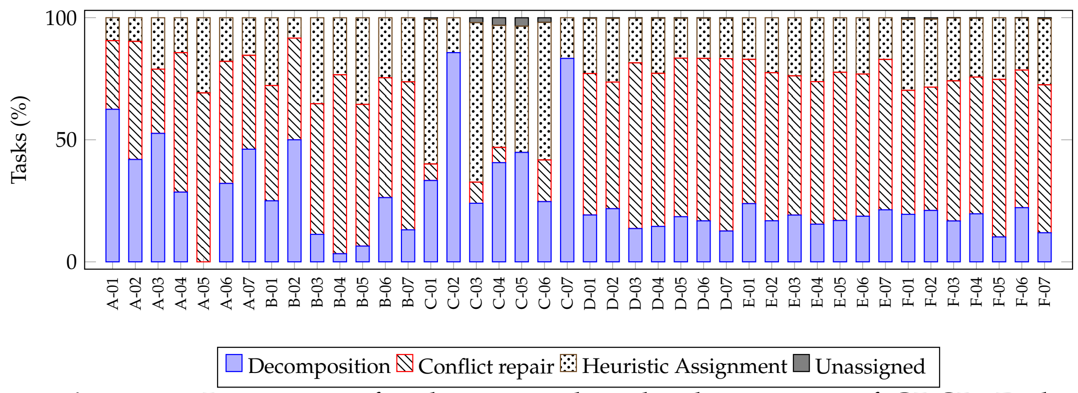


Figure 5.2: Proportion of tasks assigned in the three stages of GDCR. Each bar represents for each instance, the proportion of tasks assigned by each stage: decomposition, conflict repair and heuristic assignment. In very few cases, tasks are still left unassigned after the three stages are completed.

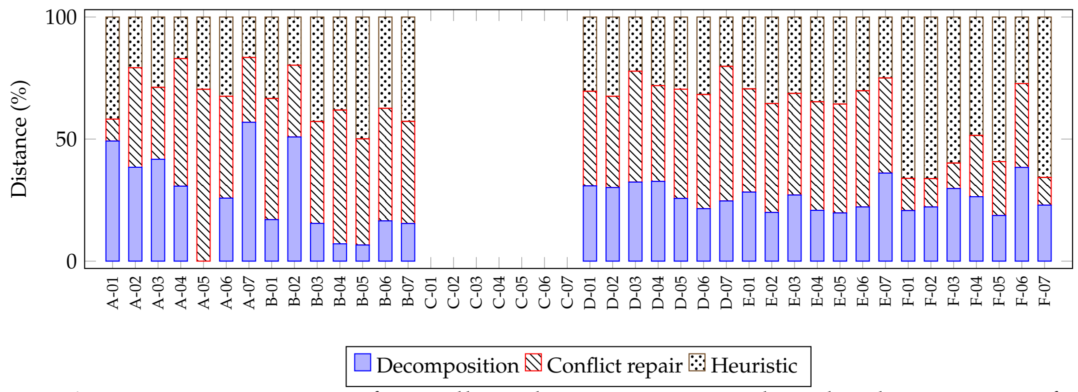


Figure 5.3: Proportion of travelling distance generated in the three stages of GDCR. Each bar represents for each instance, the proportion of travelling distance in the portion of path generated by each stage: decomposition, conflict repair and heuristic assignment.

Figure 5.3 shows the proportion of the total travelling distance corresponding from each of three stages generated to the final solution for the 42 problem instances. Note that there is no bar for C instances because no travelling between locations takes place in these solutions. Each stacked bar has three parts: decomposition, conflicting assignments repair and heuristic assignment. Each part indicates the proportion of travelling distance generated in each stage. On average, these are 26.36% for decomposition, 37.64% for conflicts repair and 36.0% for heuristic assignment. From this result, the percentage of distances by the conflict repair is the highest proportion where the proportion of

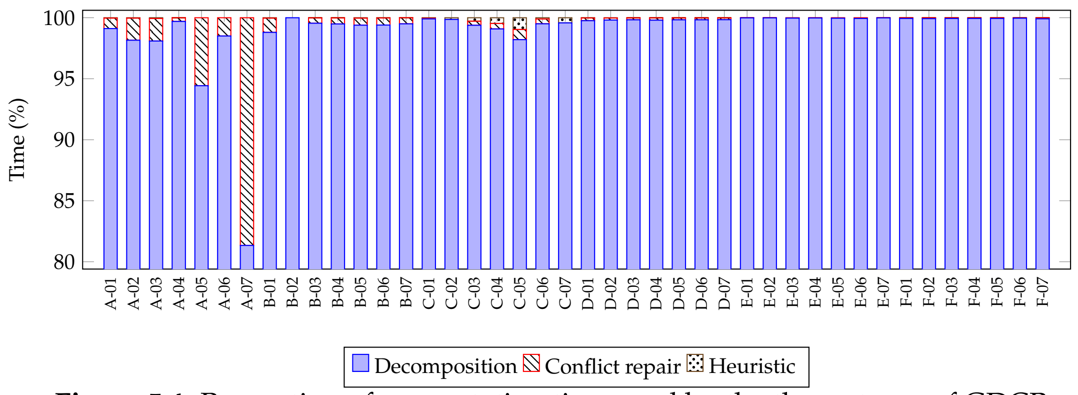


Figure 5.4: Proportion of computation time used by the three stages of GDCR. Each bar represents for each instance, the proportion of computation time used by each stage: decomposition, conflict repair and heuristic assignment.

assignments made by the conflict repair is also the highest one (47.5%). In contrast, the average proportion of distances made during the heuristic assignment is almost the same with the conflict repair distances, but the heuristic assignment stage made the lowest assignment proportion. This provides an evidence that the heuristic assignment is not as good as decomposition and repair. The stronger evidence can be seen later in this chapter that GDCA provides better solutions than an approach using only the heuristic assignment.

Figure 5.4 shows the proportion of computation time required by each of three stages for the 42 instances. Note that the y-axis starts from 80% for clearer visualisation. The larger proportion of computation time corresponds to the geographic decomposition stage as it is the only part of the method that searches considering all required visits and workforce. It is also the decomposition stage that identifies the conflicting paths to be tackled by the conflicting assignments repair stage. The heuristic assignment stage is a very quick process especially compared to the decomposition stage on the larger instances. Detail result of the actual solution computation times presented in seconds is shown as part of the experimental results in Section 5.5.

One way to shorten the computational time of the decomposition stage would be to reduce the size of the decomposition sub-problems. Our assumption is that the smaller sub-problem size will increase the number of conflicting paths and hence more conflicting sub-problems to tackle with the conflicting assignments repair stage and possibly more unassigned visits to be tackled by the heuristic assignment stage. However, we prefer to use the MIP solver as a main approach to solve the problem. Therefore, in the next section we propose a repeated decomposition and conflict repair approach.

5.4 Repeated Decomposition with Conflict Repair

RDCR is an improvement of the GDCR which aims to reduce the computational time spent in the geographical decomposition step. The main changes consist of reducing sub-problem size and introducing an iterative procedure. The process reduces stages from three to two stages: decomposition and conflicting assignments repair. The computational time can be reduced by limiting the number of visits per sub-problem to 20 visits (GDCR sets at 20 locations). Note that a location can be associated to multiple visits, therefore, the size of sub-problem is reduced. We then use decomposition and conflicting assignments repair repeatedly until no assignment can be made. This should bring higher utilization of the MIP solver instead of relying on the heuristic assignment stage.

Figure 5.5 shows an outline of the RDCR in two parts. The first part is problem decomposition presented in the upper side of the figure. The lower part of the figure presents an overview of conflicting assignments repair. These two parts are used iteratively to find an overall solution. Algorithm 8 outlines the RDCR method. The RDCR drops the heuristic assignment stage and iteratively uses the problem decomposition and conflicting assignments repair. Details of the RDCR methods are explained below.

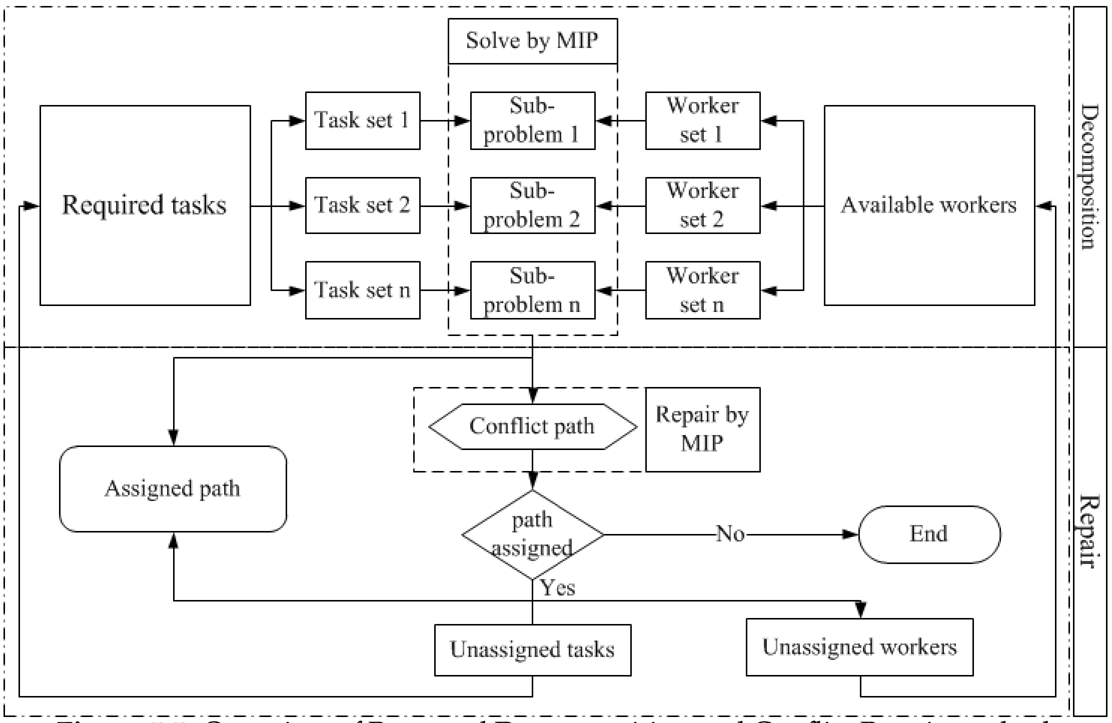


Figure 5.5: Overview of Repeated Decomposition and Conflict Repair method.

5.4.1 Problem decomposition

Problem decomposition is a main process to decrease problem size. It splits a problem into several smaller sub-problems which usually take less time to find solutions. Our problem of interest, the home healthcare problem, has two main parts available for decomposition: required visits and available workforce. Each part has its own decomposing method. Decomposing those two main parts returns a set of sub-problems where each sub-problem is small enough to tackle with the MIP solver.

Visit Partition

Visit partition is basically finding a separation rule to group several related visits together. This could fit the definition of capacitated clustering problems which clusters entities into k mutually exclusive and exhaustive group where the size of each group is restricted [96]. We apply a heuristic clustering algorithm to find the k clusters. Hence, we use local information for partitioning

Algorithm 8: Repeated Decomposition and Conflict Repair (RDCR)

```
Data: Problem $P = (K, V)$ where $K$ is a set of workers and
      $V = D \cup T \cup D'$ is a set of nodes
Result: {SolutionPaths} FinalSolution
1  {UnassignedVisits} $T'$ = $T$;
2  repeat
3      {Nodes} $V$ = $D \cup T' \cup D'$;
       /* Problem Decomposition                                  */
4      {Problem} $S$ = ProblemDecom($K$, $V$);
5      for $s \in S$ do
6          sub_sol($s$) = cplex.solve($s$);
7      end
       /* Conflicting Assignments Repair                         */
8      {Problem} $Q$ = ConflictDetection(sub_sol, $S$);
9      FinalSolution.add(NonConflictPaths(sub_sol));
10     for $q \in Q$ do
11         cRepair_sol($q$) = cplex.solve($q$);
12     end
13     FinalSolution.add(cRepair_sol);
14     $T'$ = $T$.notAssignedIn(FinalSolution);
15     Update_AvailableWorkforce($K$);
16 until No assignment made ;
```

rules such as location, geographical region, required skills and visit duration. We also use clustering algorithm k-medoids as a main clustering method.

The k-medoids algorithm works in the same way as k-means algorithm [102]. Its goal is to find k clusters based on distance between items. Therefore, the algorithm is suitable to define visit groups where their members are located within relatively smaller distances. Ideally, groups of visits should have equal sizes, which gives the minimum size of the largest sub-problem. However, using a clustering algorithm does not guarantee this since the clustering definition is to find groups of items related to their density. Hence, sub-problems may have dense area which cause some sub-problems to have a larger size.

For RDCR, we propose three variants of visit partition methods.

Location based with uniform partition (LBU) partitions visits according to their location while also aiming to limit the size of each subset. The procedure

Algorithm 9: Visit Partition: Location Based With Uniform Partition (LBU)

```
Data: {Visits} $T$, subProblemSize
Result: {{Visits}} TP = $\{T_n | n = 1, \ldots, |S|\}$; Partition set of visits
begin
    visitsList = GroupByLocation($T$);
    $n$ = 0;
    for $j \in$ visitsList do
        for $m$ = 1,...,$n$ do
            if $|T_m| <$ subproblemSize or j.shareLocation($T_m$) then
                $T_m$.add($j$);
            end
        end
        if j.isNotAllocated then
            $n = n + 1$;
            $T_n$.add($j$);
        end
    end
end
```

is shown in Algorithm 9. First, visits are ordered by location into *visitsList* and are processed one at a time. Visit *j* in *visitsList* is allocated to subset T_n if the visit has the same location as any visit already in the subset or if the maximum size of the subset has not been reached. If visit *j* is not allocated to an existing subset then a new subset is created. We set *subproblemSize* to 20 visits. Since most of the 42 HHC instances have locations with no more than 5 visits, this LBU procedure mostly generates subsets within or near the size limit.

Region based with *k*-medoids clustering algorithm (RBK) partitions visits according to geographical regions and then splits too large subsets (regions with a high density of visits) using the *k*-medoids clustering algorithm. The method basically separates visits by geographical regions, then uses a clustering algorithm to partition large regions. The clustering method to be used here is *k*-medoids clustering algorithm. The *k*-medoids clustering algorithm separates *n* visits into *k* clusters. The algorithm chooses *k* visits, each become a centre of a cluster, known as core. The other visit will become a member of the cluster where there is the smallest distance between the cluster core and the visit. The

Algorithm 10: Visit Partition: Region Based With *k*-medoids Clustering Partition (RBK)

```
Data: {Visits} $T$, subProblemSize
Result: {{Visits}} TP = $\{T_n | n = 1, \ldots, |S|\}$; Partition set of visits
begin
    {{Visits}} $A$ = firstPartition(splitVisitByRegion($T$));
    for $T_a \in A$ do
        if $|T_a| \geq$ subProblemSize then
            {{Visits}} $W$ = kMedoidCluster($T_a$,subProblemSize);
            TP.addAllSetIn($W$);
        else
            TP.add($T_a$);
        end
    end
end
```

result after apply *k*-medoids clustering algorithm is a set of subsets where visits within the same subset share the same region and are separated by short travelling distances. The procedure is shown in Algorithm 10. First, visits are partitioned by geographical regions into A and each subset T_a is processed one at a time. Then, the *k*-medoids clustering algorithm is applied to those subsets that have a size larger than *subproblemSize* (20 visits). The clustering algorithms seek to minimize travelling distance between visits in the same cluster and the clusters size is calculated by dividing the number of visits in the subset T_a by *subProblemSize*.

Skill based with *k*-medoids clustering algorithm (SBK) is a variant of RBK explained above. The only difference is that the first partitioning level is based on the skills required by visits instead of by geographical regions. Then, in Algorithm 10, we replace *splitVisitByRegion* at line 2 by *splitVisitBySkill*. The first partitioning level gives subsets with visits that require the same set of skills. This helps to group visits that may require specialist workers. Such workers with specific skills are usually low in numbers but may be require to cover visits in a wide area. The second partitioning level using *k*-medoids clustering is applied next to reduce the size of larger subsets, including those visits that require more general skills.

Workforce Selection

We propose three workforce selection methods described next, to complete the sub-problems in RDCR. The aim is to select a not too large subset of workers that are suitable for the visits already in the sub-problem.

Best Fitness Selection (BF). This procedure finds a set of best workers, where each worker is one of the best candidates for each visit in the subset. For each visit j in a subset T_a we identify the best worker by partially computing the objective function (2.14). For this, the assignment of each worker to visit j is evaluated by computing three components of the objective function: monetary cost, preferences penalty, and soft constraints penalty. The worker must also have the required skills for the visit. If the best worker identified for visit j has already been selected for another visit in the same T_a, then the next best worker is selected and so on. This selection method guarantees that all visits can be assigned unless there is no worker with the required skills for the visit. The resulting sub-problem has at most one worker for each visit.

Suppose $z(k,t)$ is a partial objective function to assign a worker k to make a visit t, K is a set of all workers and K_a is a set of available workers for sub-problem a. The BF selection procedure can be outlined in the following steps.

For each $t \in T_a$,

1. Find a worker k^* from a set K who has $z(k^*,t) = min(z(k,t))$, where $k \in K$,

2. Add the worker k^* to available worker set K_a,

3. Update the set of workers K by removing the worker k^* from set K.

Here, we can see that the best worker for a visit $t \in T_a$ is selected.

Best Average Fitness Selection (AF). This procedure finds a set of good average workers, where each worker is a good candidate for all the visits in the subset. Similar to the BF procedure, for each visit $j \in T_a$ and each worker, we partially compute the objective function (2.14). But instead of selecting the best worker for the visit, we select the $|T_a|$ best average workers, where $|T_a|$ is a number of visits in sub-problem a. Workers are listed in decreasing order of their average partial objective function value considering all visits in the subset T_n. The next available best average worker is selected for the subset until we have the same number of workers as visits in the subset.

Suppose $z(k,t)$ is a partial objective function to assign a worker k to make a visit t, K is a set of all workers, T_a is a set of visits for sub-problem a and K_a is a set of available workers for sub-problem a. The AF selection procedure can be outlined in the following steps.

1. Calculate $z(k,t)$ for every $t \in T_a$ and $k \in K$,

2. Calculate average score $z(k) = \sum_{t \in T_a} z(k,t)/|T_a|$ for each worker $k \in K$,

3. Select $|T_a|$ workers who have lowest average score $z(k)$ and add them to set K_a.

Workers Suitability Selection (WS). This procedure finds a set of suitable workers, based on skills and locations, for all the visits in the subset. All workers that have the required skills and location availability for at least one visit in the subset are selected for the subset. This selection procedure results in a larger number of workers for each sub-problem, which would demand more computational time when solving the sub-problems but could result in higher quality solutions.

Repeated Sub-problem Solving

Solving the sub-problems with the MIP solver is carried out iteratively until a final solution with a set of valid paths (with no conflicting assignments) is obtained as illustrated in Figure 5.5 and Algorithm 8. As before, the sub-problems generated with the above procedures are defined by the MIP model presented in Chapter 2. There are no conflicting assignments between the paths in the same sub-problem solution, but there might be conflicting assignments between paths in different sub-problems. Instead of using the heuristic assignment procedure as in GDCR, only the MIP solver is used in an iterative process of problem decomposition (Section 5.4.1, solving sub-problems, and conflicting assignment repair (Section 5.2.2. Noting that sub-problem solving process is to use the mathematical solver to find an optimal solution of a sub-problem.

In our experiments, smaller instances, i.e. sets A, B and C, required 2 or 3 iterations of RDCR while larger instances required 5 to 6 iterations. The first iteration was always the most time consuming and later ones (repeated repairs) were much faster. On average, the second iteration used about 20% of the first iteration computational times.

5.4.2 Experimental Study on the Sub-problem Generation Methods

We now present experimental results to investigate how the three procedures to partition visits (LBU, RBK and SBK) and the three procedures to select workforce (BF, AF and WS) contribute to generating a final solution to the whole problem instance. The nine combinations are tested on the 42 problem instances and results are collected in terms of the solution quality and computation time. In the results presented here, LBU-BF denotes location based with uniform visits partition followed by best fitness workforce. A similar naming convention is used for the other sub-problem generation procedures.

Figure 5.6 presents the summary of results comparing the nine sub-problem generation methods. From left to right, the figure shows the number of best solutions (#BestSolutions), average objective value (AverageObj) and average computational time (AverageTimes) in seconds. Each bar in each sub-figure shows the results obtained for all 42 instances when using one particular sub-problem generation method within RDCR.

In terms of number of best solutions, LBU-WS and SBK-WS achieve the highest number of best solutions (10 instances), followed by LBU-BF, RBK-BF and SBK-BF with 9 best solutions each. In terms of average objective value, eight of the methods gave very competitive results while only RBK-WS showed considerably lower performance.

In terms of average computational time, the figure seems to indicate that the LBU visits partitioning procedure combined with either BF or AF workforce selection are the fastest methods. The next fastest ones are the RBK visits partitioning procedure combined with either BF or AF. The three methods using the WS visits partitioning method are the most time consuming. As mentioned

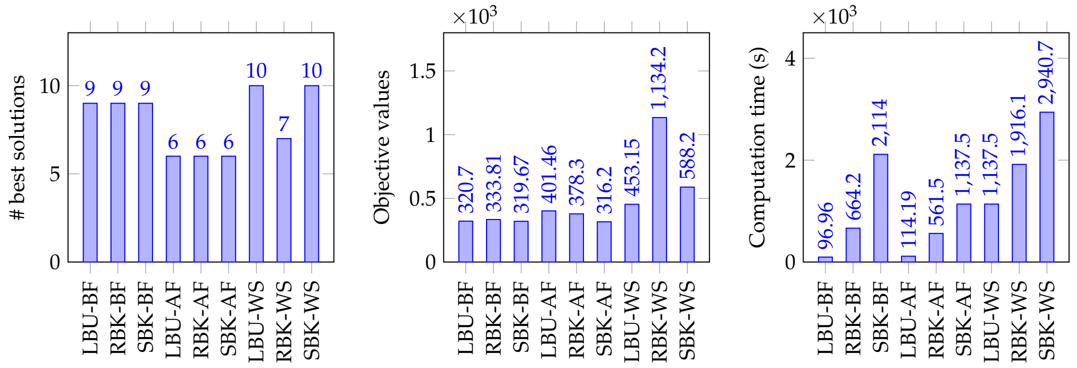


Figure 5.6: Overall results using the nine decomposition procedures within RDCR on the 42 HHC instances. The sub-figure on the left shows the number of best known solutions found with each procedure. The sub-figure in the middle shows the average objective value obtained with each procedure. The sub-figure on the right shows the average computational time in seconds when using each procedure.

Table 5.1: Friedman statistical test and mean ranks of objective value on 9 decomposition rules of RDCR. The lower mean rank presents better solution quality.

Friedman Test

N	42
χ^2	34.146
df	8
p	<.001

Mean Ranks

Workforce Selection	Task Partition		
	LBU	RBK	SBK
BF	4.44	4.50	4.31
AF	6.70	5.18	5.08
WS	4.98	5.83	3.98

before, we were expecting this to be the case as selecting all suitable workers increases the sub-problem size. However, we though that this workforce selection method would result in better solutions but this was not the case as can be seen in the other sub-figures. We should note that there was a time limit set for solving each sub-problem of 30 seconds per visit.

We also conducted a statistical analysis using the non-parametric Friedman's ANOVA test to determine any statistically significant differences, in terms of solution quality and computation time, between the sub-problem generation methods. We used SPSS [63] and set the main significance level of the test at 0.05. Based on the results of this study we selected the LBU-BF method to be used within RDCR.

Table 5.1 reports the results of this test with the calculated statistic on the left and the mean ranks on the right. The results show significant differences between the nine methods with $\chi^2(8) = 34.146, p < .001$. Therefore, we followed this with pairwise comparisons to identify differences between groups. It showed that LBU-AF produced lower solution quality (higher objective value) than the other method. Overall, the decomposition method to be used with RDCR to find the best solution quality was SBK-WS because it had the lowest objective value mean rank.

In terms of computational time, the study identified three groups, with the methods giving lower computational time being LBU-BF and LBU-AF. Table

5.2 reports the results of this test with the calculated statistic on the left and the mean ranks on the right. Statistically significant differences were found among the nine methods. Furthermore, Table 5.3 summarises the pairwise comparisons into three categories. The Positive column shows the number of other methods against which the method in the row spent more computational time with a statistically significant difference. Similarly, the Negative column shows the number of other methods against which the method in the row spent less computational time with statistically significant differences. Then, the Indifferent column shows the number of other methods against which the method did not reflect a significant difference on the computational time spent. Finally, the Category column classifies computational time of each decomposition rule into three groups: Faster, Middle, and Slower groups. In the first group are the faster methods: LBU-BF and LBU-AF. This group has two decomposition rules where they did not have positive pairwise differences. Note that more positive differences means that the rule takes higher computational time than other rules. The second group are the rules with mixed results hence in the middle of the ranking: RBK-BF, SBK-BF, RBK-AF, SBK-AF and LBU-WS. The rules in this group are slower than the faster group but still have some negative difference. Finally, there are two decomposition rules in the slower group which are RBK-WS and SBK-WS. The slower group does not have any negative differences. Hence, they require higher computational time to find a solution than other decomposition rules.

In summary, we presented a study on nine decomposition rules comparing their solution quality and computational efficiency. We applied statistical tests to find suitable decomposition rules considering on both factors. To get a high quality solution, the study showed indifference amongst the proposed rules except LBU-AF and SBK-AF. Nevertheless, the top three ranking were SBK-WS, LBU-BF and SBK-BF. The computational efficiency study presented decompos-

Table 5.2: Friedman statistical test and mean ranks of computational time on 9 decomposition rules of RDCR. The lower mean rank presents better solution quality.

Friedman Test

N	42
χ^2	161.118
df	8
p	<.001

Mean Ranks

Workforce Selection	Task Partition		
	LBU	RBK	SBK
BF	2.25	3.44	5.25
AF	3.33	3.94	5.64
WS	6.88	6.19	8.07

Table 5.3: Summation of differences in pairwise comparison between the 9 decomposition rules.

	Number of pairwise differences			Category
Decomposition rule	Negative	Indifferent	Positive	
LBU-BF	5	3	0	Faster
LBU-AF	5	3	0	
RBK-BF	3	4	1	Middle
RBK-AF	2	5	1	
SBK-BF	1	4	3	
SBK-AF	1	4	3	
LBU-WS	1	3	4	
RBK-WS	0	4	4	Slower
SBK-WS	0	2	6	

ition rules in three groups. The rules which had higher computational efficiency were LBU-BF, RBK-BF and LBU-AF. Therefore, considering both evaluating factors, the selected decomposition rule for the next study was LBU-BF as its rank was one of the top three in both evaluation factors. Based on the results of this study we selected the LBU-BF method to be used within RDCR in a comparison with the other solution methods in the next section.

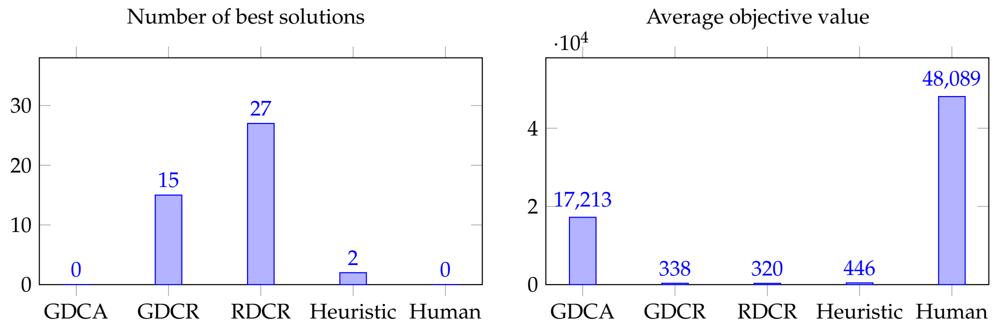


Figure 5.7: The number of best known solutions (left sub-figure) and average objective function (right sub-figure) obtained with the four algorithms and human solution (Human).

5.5 Experimental Study on the Decomposition Methods

This section presents experiments to compare the overall performance of three decomposition methods: Geographical Decomposition with Conflict Avoidance (GDCA), Geographical Decomposition with Conflict Repair (GDCR), and Repeated Decomposition and Conflict Repair (RDCR). Solutions produced by these methods are compared to solutions from the simple heuristic assignment algorithm described in Algorithm 7, solutions produced by the human planner, and the optimal solution (when available) from the MIP solver. The human planner solutions are the real-world planning solutions provided by our industrial partner.

Figure 5.7 displays two sub-graphs: the number of best solution and average objective value, provided by five solution methods. The left sub-figure shows the number of best solutions, each bar representing a solution method. The same outline displays in the right sub-figure presenting average objective value from five solution methods: GDCA, GDCR, RDCR, Heuristic and human planner.

From the result, RDCR gave the highest number of best known solutions: 27 of 42 instances. The second highest number belonged to GDCR which had 15

Table 5.4: Friedman statistical test on solution quality and computational time on five solution methods.

Objective value				Computational time			
Friedman Test		Mean Ranks		Friedman Test		Mean Ranks	
N	42	GDCA	3.79	N	42	GDCA	3.67
χ^2	136.63	GDCR	1.86	χ^2	111.71	GDCR	3.29
df	4	RDCR	1.45	df	3	RDCR	2.05
p	<.001	Heuristic	2.95	p	<.001	Heuristic	1.00
		Human	4.95				

best solutions. Heuristic provided 2 best solutions while GDCA and the human planner did not find any best solutions. Additionally, the average objective value showed similar trends as the lowest average objective value belonging to RDCR followed by GDCR, Heuristic, GDCA and human planner respectively.

Table 5.4 presents results from applying Friedman's statistical test on objective value and computational time. The test on objective value, which is presented in the left side of the table, compares five solution methods: GDCA, GDCR, RDCR, Heuristic and human planner. The computational time, presented in the right side of the table, shows comparison amongst four solution methods as solutions from human planner were calculated manually where the computation time is unknown.

The Friedman ANOVA test on objective value shows significant difference in solution quality between five methods with $\chi^2(4) = 136.63, p < .001$. Again, pairwise comparisons had been used which showed almost all pairs of algorithm produced statistical significant different results. The exception were the pairs RDCR:GDCR and Heuristic:GDCA. Hence, it can be concluded that the best methods judging by solution quality were RDCR and GDCR.

The objective values by instances are presented in Table 5.5. It displays objective values of six methods: GDCA, GDCR, RDCR, heuristic algorithm, human planner (Human), and optimal solution by the MIP solver when available. Optimal solutions are available only for instance set WSRP-A, WSRP-

Table 5.5: Objective value obtained for each of 42 problem instances by solving a problem as a whole (Optimal), the GDCR method, the GDCA method and baseline heuristic algorithm.

Set	Opt	GDCA	GDCR	RDCR	Heur	Human
A-01	**3.49**	5.65	4.48	4.00	5.46	307
A-02	**2.49**	4.53	3.36	2.93	4.42	75.3
A-03	**3.00**	10.6	4.93	6.86	7.41	68.4
A-04	**1.42**	3.09	2.49	2.48	2.36	93.2
A-05	**2.42**	3.54	3.12	**2.42**	3.15	24.5
A-06	**3.55**	3.74	3.62	3.56	5.67	24.6
A-07	**3.71**	4.81	4.07	**3.71**	7.10	27.7
B-01	**1.70**	1.79	1.89	1.76	2.87	200
B-02	**1.75**	1.89	**1.75**	1.80	2.83	1.94
B-03	**1.72**	2.06	1.89	1.85	2.97	692
B-04	**2.07**	2.21	2.13	2.12	3.01	130
B-05	**1.82**	4.74	2.54	2.98	2.88	623
B-06	**1.62**	2.52	1.75	1.75	2.91	112
B-07	**1.79**	4.06	2.94	2.03	3.44	474
C-01	N/K	905	133	**132**	185	29,642
C-02	**3.15**	3.61	**3.15**	**3.15**	4.86	6.41
C-03	N/K	1,186	196	**159**	170	24,295
C-04	**11.15**	81.3	23.1	13.1	16.5	997
C-05	**12.34**	68.9	22.4	15.3	17.6	752
C-06	N/K	3,102	198	**197**	251	11,486
C-07	**4.30**	5.29	**4.30**	**4.30**	5.82	10.7
D-01	N/K	496	210	**205**	236	17,386
D-02	N/K	373	206	**199**	244	13,830
D-03	N/K	3,213	229	**208**	221	26,919
D-04	N/K	419	219	**212**	223	16,677
D-05	N/K	244	202	**184**	189	33,705
D-06	N/K	1,411	223	**199**	**199**	23,869
D-07	N/K	753	218	202	**197**	22,794
E-01	N/K	33.0	**3.69**	5.19	6.27	180,633
E-02	N/K	26.0	**2.21**	3.22	4.83	78,012
E-03	N/K	29.0	**1.23**	4.23	8.27	61,624
E-04	N/K	28.5	**1.79**	**1.79**	4.30	101,369
E-05	N/K	270	**3.76**	7.26	8.25	32,075
E-06	N/K	24.6	**2.30**	**2.30**	5.43	80,479
E-07	N/K	428	**4.72**	7.71	5.68	142,485
F-01	N/K	64,305	2,740	**2,150**	2,810	89,383
F-02	N/K	73,291	**2,482**	2,505	3,235	117,274
F-03	N/K	115,235	707	**704**	1,619	141,427
F-04	N/K	102,994	1,453	**1,448**	1,958	110,104
F-05	N/K	101,438	**297**	315	1,752	336,684
F-06	N/K	76,007	747	**742**	862	146,456
F-07	N/K	176,541	3,610	**3,604**	4,239	176,524

Bold text refers to the best solution.
N/K is for solution currently not known.

B and some of set WSRP-C. The table shows differences on solution quality between decomposition methods and optimal solution. Only GDCR and RDCR had reached some of the optimal solutions: B-02, C-02 and C-07 for GDCR and A-05, A-07, C-02 and C-07 for RDCR.

The computation time used by each of the five methods was recorded and presented in Table 5.6. Note that time spent by the human planner cannot be displayed here as it was processed manually. As expected, the computation time spent in finding the optimal solution grows quickly as the problem size increases. This can be appreciated by comparing the corresponding columns for the instance groups WSRP-A and WSRP-B. The optimal solution could not be found for most problem instances and this is marked as N/K in the table. On average, GDCA used the highest computational time followed by GDCR, RDCR and Heuristic algorithm. From the data table, it was clear that the heuristic algorithm was the quickest method. This is confirmed by statistical test, whereby Friedman's ANOVA showed statistically significant differences between the computational times of four methods with $\chi^2(3) = 111.717, p < .001$. Pairwise comparisons showed they had significant difference between all four methods except between GDCR and GDCA. The results confirm that the heuristic algorithm spent the least computational time. Furthermore, RDCR was the quickest among decomposition methods.

Therefore, by considering both solution quality and computational efficiency, we can say that RDCR was the best option as it had the best quality solution and was ranked second in computational time.

Table 5.6: Computation time (seconds) obtained for each of 42 problem instances by solving a problem as a whole (Optimal), GDCA, GDCR, RDCR, and simple heuristic assignment.

Set	Opt	GDCA	GDCR	RDCR	Heur
A-01	7.67	3.71	3.76	2.53*	**<.1**
A-02	7.07	3.58	3.35	2.39*	**<.1**
A-03	12.6	3.70*	4.69	5.17	**<.1**
A-04	5.22	2.88	2.29	1.87*	**<.1**
A-05	1.38	1.77	1.28	0.76*	**<.1**
A-06	4.90	2.42	2.80	1.77*	**<.1**
A-07	1.54	1.64	1.55	0.70*	**<.1**
B-01	21.2	8.07	6.96	4.67*	**<.1**
B-02	2.14	4.29	3.36	0.79*	**<.1**
B-03	6,003	32.86	37.97	10.51*	**<.1**
B-04	21.4	15.25	12.19	2.63*	**<.1**
B-05	585	25.35	23.22	8.31*	**<.1**
B-06	184	24.11	21.80	8.78*	**<.1**
B-07	300	23.64	24.44	8.14*	**<.1**
C-01	N/K	211	224	25.50*	**0.34**
C-02	6	0.57	0.63	0.12*	**<.1**
C-03	N/K	26.33	27.84	18.22*	**0.6**
C-04	90	3.09	3.84	1.07*	**0.11**
C-05	55	1.05	1.91	0.71*	**<.1**
C-06	N/K	47.05	49.77	24.94*	**0.19**
C-07	1	0.24	0.23	0.11*	**<.1**
D-01	N/K	1,060	579	109*	**0.18**
D-02	N/K	1,192	706	109*	**0.14**
D-03	N/K	1,209	1,024	127*	**0.18**
D-04	N/K	3,005	785	127*	**0.17**
D-05	N/K	1,307	907	118*	**0.18**
D-06	N/K	1,222	1,064	130*	**0.2**
D-07	N/K	1,362	1,133	142*	**0.23**
E-01	N/K	8,408	7,676	93.51*	**0.19**
E-02	N/K	12,448	9,806	86.84*	**0.18**
E-03	N/K	20,747	11,872	101*	**0.22**
E-04	N/K	15,190	8,758	71.57*	**0.18**
E-05	N/K	32,619	9,510	98.63*	**0.25**
E-06	N/K	24,212	9,121	65.60*	**0.15**
E-07	N/K	51,057	13,884	107*	**0.27**
F-01	N/K	3,446	1,788	250*	**1.00**
F-02	N/K	1,111	1,730	251*	**1.20**
F-03	N/K	4,555	1,908	342*	**1.61**
F-04	N/K	4,219	7,060	360*	**1.67**
F-05	N/K	6,157	3,437	390*	**1.91**
F-06	N/K	9,696	7,204	442*	**1.91**
F-07	N/K	3,833	1,847	422*	**2.46**

Bold text refers to the lowest computational time.
N/K is for solution currently not known.
* the second fastest computational time.

5.6 Conclusion

We have presented two decomposition methods based on conflict repairing to improve the overall performance of the method Geographical Decomposition with Conflict Avoidance (GDCA). The two methods described in this chapter were Geographical Decomposition with Conflict Repair (GDCR) and Repeated Decomposition and Conflict Repair (RDCR). The conflicting assignments repair was proposed to fix the main weak point of GDCA which was the conflict avoidance process. The conflict avoidance needed a sub-problem solving order which gave full resources to the first sub-problem in the solving queue while the other sub-problems had restricted resources. Conflict repair approaches, on the other hand, did not have a problem-solving order as it allowed all sub-problems to use all resources without considering conflicting visits between sub-problems. Conflicting assignments were tackled during the conflict repair process.

The GDCR had shown improvement from GDCA as presented in its solutions. This study also presented the contributions on a solution provided by three stages of GDCR. The result showed that the conflicting assignments repair stage made the most assignments. It also showed that the geographical decomposition stage spent the most computational time. Therefore, RDCR was proposed to reduce the overall computational time by shortening the time spent by the decomposition process. The changes were reducing the size of decomposition sub-problems and introducing an iterative process between decomposition and conflicting assignments repair. Additionally, the experiment applied nine decomposition rules expecting to find the best decomposition rule for RDCR. Result showed that partitioning visits by location based on uniform partition and selecting workforce by the best fitness selection (LBU-BF) is the overall best decomposition rule when measured by solution quality and computational time usage. Nevertheless, comparing RDCR using LBU-BF with the GDCR shows improvement on both solution quality and computational time. Therefore, we can say that the iterative process gives solution quality improvement which more than compensates for the effect of reducing the sub-problem size.

The study also compared decomposition techniques with other solution methods which were solving a problem as a whole by the MIP solver, the simple heuristic assignment algorithm and the human planner. Heuristic method, on the other hand, produced solutions with no difference in quality compared to GDCA but the heuristic algorithm spent the least computational time. The human planner solution was presented basically to show how an automated system should improve solutions if it is deployed to normal practice. It was shown that solutions by automated methods produced schedules of better quality. The experimental results also showed that the method providing the best solution quality so far is the RDCR and the GDCR is the second best on both the solution quality and the computational time.

The research should continue to improve the decomposition methods to gain both solution quality and computational efficiency as they still have room for improvement. In particular, RDCR should gain a significant improvement to computational time when applying parallel computing as multiple sub-problems can be tackled at the same time.

Chapter 6

Repeated Decomposition and Conflict Repair on other Benchmark Workforce Scheduling and Routing Problems

This chapter applies a heuristic decomposition method, the Repeated Decomposition and Conflict Repair (RDCR) to the WSRP with time-dependent activities constraints. Mathematical formulations describing the WSRP with time-dependent activities was proposed by Rasmussen et al. [107].

The content of this chapter has been presented in

- Wasakorn Laesanklang, Dario Landa-Silva and J. Arturo Castillo-Salazar. **Mixed Integer Programming with Decomposition for Workforce Scheduling and Routing With Time-dependent Activities Constraints.** In *Proceedings of the 5th International Conference on Operations Research and Enterprise Systems (ICORES 2016)*, pp. 283–293, Scitepress, Rome, Italy, February 2016.

- Wasakorn Laesanklang, Dario Landa-Silva and J. Arturo Castillo-Salazar.

An Investigation of Heuristic Decomposition to Tackle Workforce Scheduling and Routing With Time-dependent Activities Constraints. submitted, to be appear in *Operations Research and Enterprise Systems*, Series Communications in Computer and Information Science.

6.1 Problem Description and Formulation

This section describes the workforce scheduling and routing problem with time-dependent activities constraints and the mixed integer programming (MIP) model used to formulate this problem. The MIP model was originally presented in [107] for a home care crew scheduling scenario. This scenario and several others are tackled here with the solution technique proposed in this chapter. The type of WSRP tackled here is one involving time-dependent activities constraints, i.e. situations in which visits relate to each other time-wise. More detail on instances and their features are summarised in Section 6.3.1. This section focuses on describing the problem constraints arising in such scenarios and their formulation.

6.1.1 Mixed Integer Programming Model for Workforce Scheduling and Routing Problem with Time-dependent Activities Constraints

The MIP model presented in Chapter 2 has been used to tackle the 42 HHC instances. The WSRP instances in this chapter have slightly different constraints. The additional constraints in this chapter are time-dependent activities constraints (see Section 2.3.9) and the constraints that are not required in these WSRP instances are workforce time availability constraint (2.11), (2.12) and workforce region availability constraint (2.13). The problem definition of the

WSRP with time-dependent activities constraints is the same with a home care crew scheduling problem tackled by Rasmussen et al. [107]. The same problem had been tackled by Castillo-Salazar et al. [36].

Notation used in this MIP model is the same as that which was presented in Chapter 2. We repeat the notation definition in Table 6.1. We emphasise the parameter $s_{i,j}$ which plays important roles in time-dependent activities constraints. The value for this parameter is varied subject to constraint types, more information can be found in Section 6.1.2.

The objective function of the model has been reduced to three tiers because the soft violation penalties are no longer needed, where the objective function is presented in (6.1). The first cost is the *monetary cost* (denoted $c^k_{i,j}$) which is the cost of assigning each worker k to visit i and then move to the location of visit j. The weight correspondent to this cost is λ_1. The second main cost is the *preferences cost* (denoted ρ^k_i) which is the cost of assigning a lower preference worker to a visit, i.e. not assigning the most preferred worker to a given visit. The correspondent weight is λ_2. The third main cost is the *unassigned visits* which the cost added when a decision variable $y_j = 1$. The weight of the cost is λ_3. The level of priority to each cost is controlled by the weights λ_1, λ_2, and λ_3. The values for these weights are set in the same way with Castillo-Salazar et al. [36] so that this result of this study can be compared with the algorithm proposed by Castillo-Salazar et al. [36].

Finally, the summarised description of the MIP model is in the following constraints: a visit is either assigned to workers or left unassigned (6.2). It can only be assigned to workers who are qualified to undertake activities associated to the visit (6.3). Each path must start from the worker's initial location (6.4) and end at the final location (6.5). The flow conservation constraint guarantees that once worker k arrives to a visit location, then leaving that location occurs in order to form a working path (6.6). Another constraint is that the visit

Table 6.1: Notation used in MIP model for WSRP

Sets	Definition
V	Set of all nodes denoted by $V = D \cup T \cup D'$. Indices $i, j \in V$ instantiate nodes.
D	Set of source nodes, i.e. starting locations.
D'	Set of sink nodes, i.e. ending locations.
T	Set of visiting nodes.
V^S	Set of nodes have leaving edges, i.e. $V^S = D \cup T$.
V^N	Set of nodes have entering edges, i.e. $V^N = D' \cup T$.
E	Set of edges connecting two nodes.
K	Set of workers, k is a worker in K.
S	Set of dependency visits. Members are pairs of visit (i, j) in which visit i and j are dependent.
Parameters	
M	Large constant.
$\lambda_1, \ldots, \lambda_4$	Objective weights.
$t_{i,j} \in \mathbb{R}^+$	Travelling duration between node $i \in V^S$ and node $j \in V^N$.
$d_{i,j} \in \mathbb{R}^+$	Travelling distance between node $i \in V^S$ and node $j \in V^N$.
$p_j^k \in \mathbb{R}^+$	Costs of assigning worker k to node $j \in T$.
$\rho_j^k \in \mathbb{R}^+$	Preferences value of assigning worker k to node $j \in T$.
$r_j \in \mathbb{N}$	The number of required workers at node $j \in T$.
$\delta_j \in \mathbb{R}^+$	Duration of visit at node $j \in T$.
$\alpha_L^k, \alpha_U^k \in \mathbb{R}^+$	Shift starting and ending time for worker k.
$w_j^L, w_j^U \in \mathbb{R}^+$	Lower and upper time windows to arrive node j.
$h^k \in \mathbb{R}$	Maximum working duration for worker k.
$\eta_j^k \in \{0, 1\}$	Qualification of worker k at node j, the value is 1 when a worker is qualified to work, 0 otherwise.
$\gamma_j^k \in \{0, 1\}$	Worker region availability on node j, the value is 1 when a worker is available in the region of visit j, 0 otherwise.
$s_{i,j} \in \mathbb{R}$	Dependency coefficient. The value states relation of visit i and visit j when $(i, j) \in S$.
$Q^k \in \mathbb{R}$	Skill proficiency levels of worker $k \in K$
$q^j \in \mathbb{R}$	Minimum qualification levels required to make a visit $j \in T$
Variables	
$x_{i,j}^k \in \{0, 1\}$	Worker assignment decision variable, the value is 1 when a link between $i \in V^S$ and $j \in V^N$ is assigned to worker k, 0 otherwise.
$\omega_j \in \{0, 1\}$	Working shift violation indicator variable, the value is 1 when the assignment at node j is made outside working shift, 0 otherwise.
$\psi_j \in \{0, 1\}$	Worker's region violation indicator variable, the value is 1 when the assignment at node j is violated, 0 otherwise.
$y_j \in \mathbb{N}$	Unassigned visit indicator variable, the value is 1 when assignment does not make at node j.
$a_j^k \in \mathbb{R}^+$	Arrival time decision variable for worker k to work at node j.

$$\text{Minimise}\quad \lambda_1 \sum_{k\in K}\sum_{i\in V^S}\sum_{j\in V^N} c^k_{i,j}x^k_{i,j} + \lambda_2 \sum_{k\in K}\sum_{i\in T}\sum_{j\in V^N} \delta^k_i x^k_{i,j} + \lambda_3 \sum_{i\in T}\gamma_i y_i \tag{6.1}$$

Subject to

$$\sum_{k\in K}\sum_{i\in V^S} x^k_{i,j} + y_j = 1 \quad \forall j\in T \tag{6.2}$$

$$\sum_{j\in V^S} x^k_{i,j} \le \rho^k_j \quad \forall k\in K, \forall j\in T \tag{6.3}$$

$$\sum_{j\in V^N} x^k_{d^k,j} = 1 \quad \forall k\in K \tag{6.4}$$

$$\sum_{i\in V^S} x^k_{i,d^k} = 1 \quad \forall k\in K \tag{6.5}$$

$$\sum_{i\in V^S} x^k_{i,h} - \sum_{j\in V^N} x^k_{h,j} = 0 \quad \forall k\in K, \forall h\in T \tag{6.6}$$

$$w^L_j \sum_{i\in V^S} x^k_{i,j} \le a^k_j \quad \forall k\in K, \forall j\in V^N \tag{6.7}$$

$$a^k_j \le w^U_j \sum_{j\in V^N} x^k_{i,j} \quad \forall k\in K, \forall j\in V^N \tag{6.8}$$

$$\alpha^L_k \le a^k_j \quad \forall k\in K, \forall j\in T \tag{6.9}$$

$$a^k_j + \delta_j \le \alpha^U_k \quad \forall k\in K, \forall j\in T \tag{6.10}$$

$$a^k_i + s^k_{i,j}x^k_{i,j} \le a^k_j + w^U_i(1 - x^k_{i,j}) \forall k\in K, \forall i\in V^S, \forall j\in V^N \tag{6.11}$$

$$w^L_i y_i + \sum_{k\in K} a^k_i + s_{i,j} \le \sum_{k\in K} a^k_j + w^U_j y_j \quad \forall i,j\in S \tag{6.12}$$

$$x^k_{i,j} \quad \text{are binary,} \quad \forall k\in K, \forall i\in V^S, \forall j\in V^N \tag{6.13}$$

$$y_j \quad \text{are binary,} \quad \forall j\in T \tag{6.14}$$

$$a^k_j \ge 0 \quad \forall k\in K, \forall j\in V \tag{6.15}$$

associated must start in the given time window as denoted by (6.7) and (6.8). Assignments of visits to workers must respect the worker's availability, (6.9) and (6.10). The time allocated for starting a visit must respect the travel time needed after completing the previous visit (6.11). The time-dependent activities constraints (6.12) enforce arrival times of the time-dependent visits, more details of these constraints will be presented in Section 6.1.2. The methodology presented in this paper has been adapted to tackle this type of constraints in particular. Lastly, the types of decision variables in this MIP model are specified by (6.13), (6.14) and (6.15).

6.1.2 Time-dependent Activities Constraints

A key difference of the WSRP tackled in this chapter and the HHC problem explained in Chapter 2 is that the WSRPs include a special set of constraints called *time-dependent activities constraints* that establish some inter-dependence between activities as denoted by (6.12). These constraints reduce the flexibility in the assignment of visits to workers because, for example, a pair of visits might need to be executed in a given order. There are five constraint types: *overlapping*, *synchronisation*, *minimum difference*, *maximum difference* and *minimum-maximum difference*. Table 6.3 shows the value given to the time-dependent parameter in constraint (6.12) for each type of time-dependent activity constraint. Table 6.4 presents the formulation for each of these constraints when $s_{i,j}$ has been applied. A solution that does not comply with the satisfaction of these *time-dependent activities constraints* as defined in Table 6.4 is considered infeasible.

- *Overlapping* constraint means that the duration of one visit i must extend (partially or entirely) over the duration of another visit j. This constraint is satisfied if the end time of visit i is later than the start time of visit j and also the end time of visit j is later than the start time of visit i. Therefore, $s_{i,j} = -\delta_j$ and $s_{j,i} = -\delta_i$.

- *Synchronisation* constraint means that two visits must start at the same time. This constraint is satisfied when the start times of visits i and j are the same. Therefore, $s_{i,j} = s_{j,i} = 0$.
- *Minimum difference* constraint means that there should be a minimum time between the start time of two visits. This constraint is satisfied when visit j starts at least s_i^l time units after the start time of visit i. Therefore, $s_{i,j} = s_i^l$.
- *Maximum difference* constraint means that there should be a maximum time between the start time of two visits. This constraint is satisfied when visit j starts at most s_i^u time units after the start time of visit i. Therefore, $s_{j,i} = -s_i^u$.
- *Minimum-maximum difference* constraint is a combination of the two previous conditions and it is satisfied when visit j starts at least s_i^l time units but not later than s_i^u time units after the start time of visit i. Therefore, $s_{i,j} = s_i^l$ and $s_{j,i} = -s_i^u$.

Table 6.2: Notations and definition for constraint (6.12)

Notation	Definition
$(i, j) \in S$	i, j is a pair of visits with time dependency and both assigned in a solution.
$a_i^{k_1}, a_j^{k_2}$	The start times for visit i and j assigned to employees k_1 and k_2 respectively.
δ_i, δ_j	The durations of visit i and visit j respectively.
s_i^l, s_i^u	Minimum difference and maximum difference duration respectively between visit i and visit j.

Table 6.3: Value of time-dependent parameter $s_{i,j}$ (constraint 6.12) for each of the five time-dependent activities constraints.

Constraint Types	$s_{i,j}$	$s_{j,i}$
Overlapping	$-\delta_j$	$-\delta_i$
Synchronisation	0	0
Minimum difference	s_i^l	N/A
Maximum difference	N/A	$-s_i^u$
Minimum-maximum difference	s_i^l	$-s_i^u$

Table 6.4: Conditions to validate the satisfaction of each time-dependent activities constraint.

Constraint Types	Validate Condition
Overlapping	$a_i^{k_1} + \delta_i \geq a_j^{k_2}$ $a_j^{k_2} + \delta_j \geq a_i^{k_1}$
Synchronisation	$a_i^{k_1} = a_j^{k_2}$
Minimum Difference	$a_i^{k_1} + s_i^l \leq a_j^{k_2}$
Maximum Difference	$a_i^{k_1} + s_i^u \geq a_j^{k_2}$
Minimum-Maximum Difference	$a_i^{k_1} + s_i^l \leq a_j^{k_2}$ $a_i^{k_1} + s_i^u \geq a_j^{k_2}$

6.2 Time-Dependent Activities Constraint Modification to the Repeated Decomposition and Conflict Repair Method

We deploy RDCR, which was presented in Chapter 5, to solve the WSRP with time-dependent activities constraints. We choose LBU-BF for its decomposition rule because the study showed the best results among the rules. Although, there are modifications needed in the problem decomposition stage and the conflicting assignment repair stage, these modifications are mainly to accommodate time-dependent activities constraints.

6.2.1 Modification in Problem Decomposition Stage

We remind the reader that problem decomposition is a stage which has three parts: visit partition, workforce selection and sub-problem solving. For this problem, every sub-problem is defined by formulations (6.1) to (6.15) presented in this chapter. The modification made on the problem decomposition is to focus on visit partition because the time-dependent activities constraints are defined for a pair of visits.

Algorithm 11 shows the steps for the modified Location Based with Uniform Partition (LBU). The modified LBU works in a similar way as the LBU presented in Chapter 5. The only modification is that if the algorithm finds a visit which has a time-dependent pair, the algorithm adds both visits in the same subset, as shown in line 8 of Algorithm 11.

The modification guarantees that assignment made by problem decomposition respect time-dependent activities constraints. Although, solutions to the sub-problem solved in the problem decomposition stage could have conflicting assignments which need to be repaired, the conflicting assignment repair

Algorithm 11: Modified Location Based With Uniform Partition (LBU)

```
Data: {Visits} $T$, subProblemSize
Result: {{Visits}} $TP = \{T_i | i = 1, \ldots, |S|\}$; Partition set of visits
visitsList = OrderByLocation($T$);
$n$ = 0;
for $j \in visitsList$ do
    for $m$ = 1,...,$m$ do
        if $|T_m| <$ subproblemSize or j.shareLocation($T_m$) then
            $T_m$.add($j$);
            if j.hasTimeDependent then
                Visit $i$ = PairedVisit($j$);
                $T_m$.add($i$);
            end
        end
    end
    if j.isNotAllocated then
        $n$=$n$+1;
        $T_n$.add($j$);
        if j.hasTimeDependent then
            Visit $i$ = PairedVisit($j$);
            $T_n$.add($i$);
        end
    end
end
```

fixes the conflicting assignments by defining conflicting sub-problems in which each sub-problem contains a worker and their visits that were on the set of conflicting paths. A conflicting sub-problem is then being solved individually. At this stage, we can see that the new assignments might be rearranged and time-dependent activities might not be guaranteed. This could result in time-dependent activities constraints being violated in the conflicting assignment repair.

Hence, we propose a modification to conflicting assignment repair. The approach mainly keeps the assignment time of time-dependent assignments provided by the solutions of the problem decomposition stage which satisfy time-dependent activities constraints. Thus, for every time-dependent visit, the process sets time window $w_i^L = w_i^U = a_i$ where i is a time dependent visit and

a_i is an arrival time at visit i given by the problem decomposition stage. This step is deployed before generating the conflicting sub-problems. Once the fixed time restriction is enforced, it affects every iteration of the process.

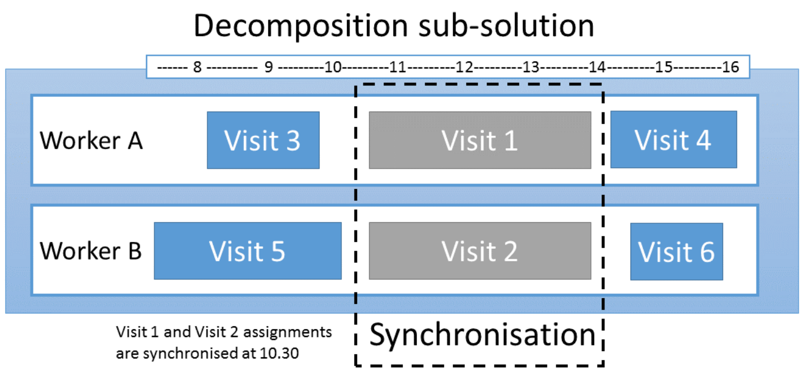
We present four possible cases in the time-dependent activities constraint modification.

1. **Assignments in the solution obtained from solving a decomposition sub-problem do not require conflicting assignment repair.**

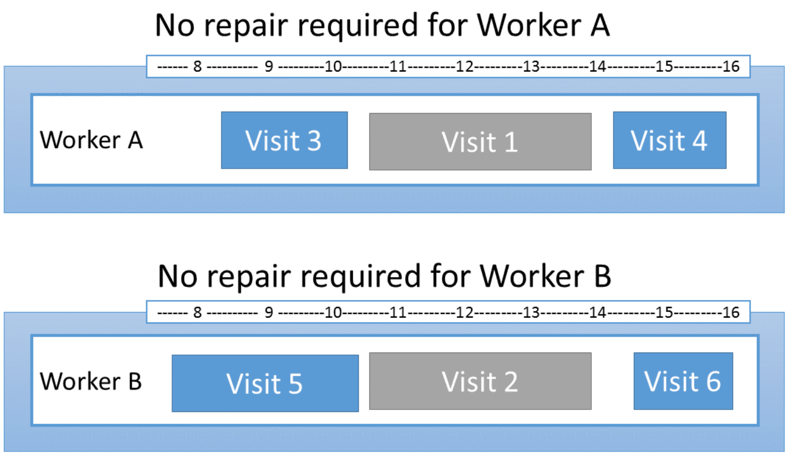
   The solution obtained from solving a decomposition sub-problem satisfies the sub-problem constraints. If workers have not been used in the other sub-problem solutions, the paths of these workers satisfy the constraints of the full problem. The paths also satisfy the time-dependent activities constraints because the constraints have been defined in the sub-problem model and both visits are in the same sub-problem.

   Figure 6.1 illustrates an example of this case where the synchronisation constraint takes place. The figure contains two sub-figures of which one illustrates a decomposition sub-problem solution and the other presents the paths that will use in the final solution. For this example, the synchronisation constraint is enforced, where Visit 1 and Visit 2 must be synchronised. Therefore, Visit 1 and Visit 2 must be grouped in the same sub-problem.

   After the sub-problem solving steps, the MIP solver produces a sub-problem solution, as illustrated in Sub-figure 6.1a, where Worker A is assigned to make Visit 3, Visit 1, and Visit 4 and Worker 2 is assigned to make Visit 5, Visit 2, and Visit 6. Assignments of Visit 1 and Visit 2 are synchronised where their starting times are both set at 10.30. By assumption of this example, paths of Worker A and Worker B are not required to be repaired. Thus, they can be used in the final solution and both paths satisfy



(a) Decomposition sub-problem solution



(b) Conflict Repair sub-problems

Figure 6.1: Illustration of time-dependent constraint modification example when the assignments do not need conflicting assignments repair. Sub-figure (a) shows solution from solving a decomposition sub-problem where a synchronisation constraint has been enforced. Sub-figure (b) presents paths of two workers which have synchronised visits. Assumption is that the two workers are only been used in one decomposition sub-solution. The two paths can be used directly in the final solution.

all constraints in the full model, as illustrated in Sub-figure 6.1b.

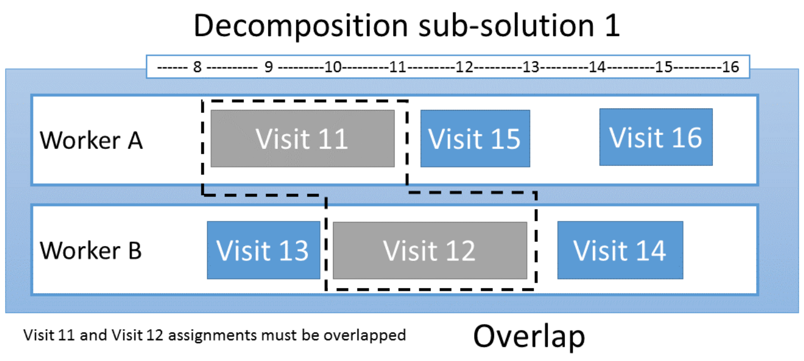
2. **Assignments in the solution obtained from solving a decomposition sub-problem need to be repaired and the time-dependent activities have been assigned by the conflicting assignments repair.**

   This case is applied when a worker has been used in two or more decomposition solutions where each solution has a path for that worker.

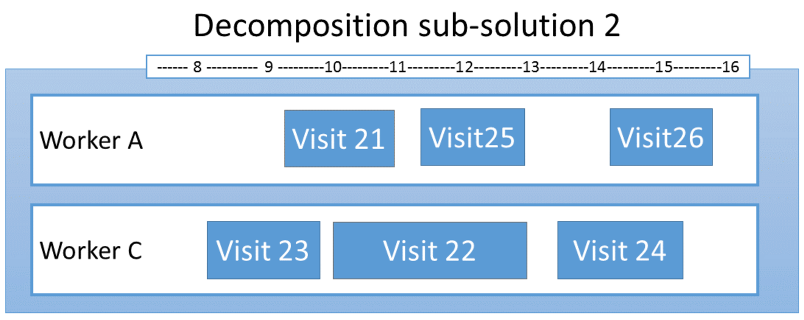
   Figure 6.2 illustrates an example of this case with an overlapping constraint. The figure has three sub-figures: two sub-figures show two decomposition sub-problem solutions, and the other sub-figure presents conflicting assignments repair sub-problem and its solution path. For this example, we assume that Visit 11 and Visit 12 are dependent by an overlapping constraint. Thus, sub-problem 1 is built where the two time-dependent visits are grouped.

   The sub-problem 1 is solved by the MIP solver where the solution can be illustrated in Sub-figure 6.2a. From the sub-figure, Visit 11 and Visit 12 are overlapped as Visit 11 starts at 8.10 and ends at 11.00 and Visit 12 starts at 10.00 and ends at 13.00. The Visit 11 is assigned to Worker A and the Visit 12 is assigned to Worker B. At the same stage, Worker A is used in the solution for sub-problem 2, as shown in Sub-figure 6.2b. As a result, Worker A has been assigned to two working paths. For this example, we assume that Worker B has not been used in the other sub-problem solutions except the solution for sub-problem 1, thus the path for Worker B is passed to the final solution.

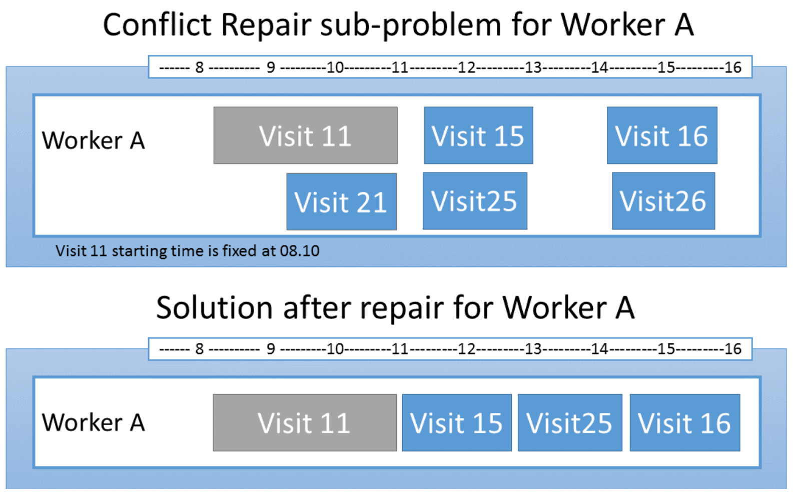
   For Worker A, combining two paths will result in conflict assignments as shown in the conflicting repair sub-problem for Worker A in sub-figure 6.2c. The conflicting assignment repair is considered as a new sub-problem to be solved by the MIP solver. However, to maintain the overlapping



(a) Decomposition sub-problem 1 solution



(b) Decomposition sub-problem 2 solution



(c) Conflict Repair sub-problems

Figure 6.2: Illustration of time-dependent constraint modification example when conflicting assignments repair assigns time-dependent activities. Sub-figure (a) shows solution from solving decomposition sub-problem 1 where an overlapping constraint has been enforce. Sub-figure (b) presents solution from solving decomposition sub-problem 2. Worker A has been used in two decomposition sub-problems. Sub-figure (c) presents conflict repair sub-problem for Worker A where Visit 11 has a fixed assignment time at 08.10 and the solution after repair for the Worker A where Visit 11 overlaps with Visit 12.

constraint, which enforced between Visit 11 and Visit 12, the modification has been made to the Visit 11 time window by enforcing a fixed starting time at 8.10.

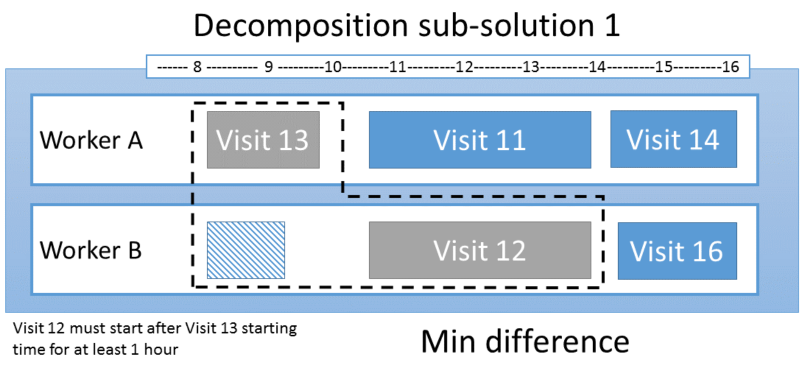
As a result, the conflicting assignments repair assigns the Visit 11 to the Worker A as illustrated by the solution after repair in Sub-figure 6.2c. The visit assignment in the solution after repair remains overlapped with Visit 12. From the same figure, we can see that the assigned times of Visit 15 and Visit 16 are changed to have Visit 25 assigned. The modification works in the same way with this example, when both time-dependent visits are repaired.

3. **Assignments in the solution obtained from solving a decomposition sub-problem need to be repaired and the time-dependent activities were not assigned by the conflicting assignments repair.**

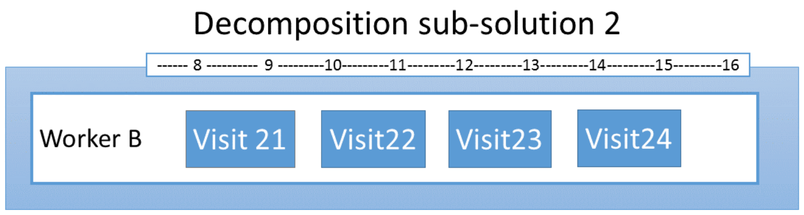
   This case is applied when a worker has been used in two or more decomposition solutions where each solution has a path for that worker.

   Figure 6.3 illustrates an example of this case with a minimum difference constraint. The figure has three sub-figures: two of them shows two decomposition sub-problem solutions, and the other sub-figures presents conflicting assignment repair sub-problem and its solution. For this example, Visit 12 and Visit 13 are time-wise dependent where Visit 12 must take place at least 1 hour after the Visit 13 starting times.

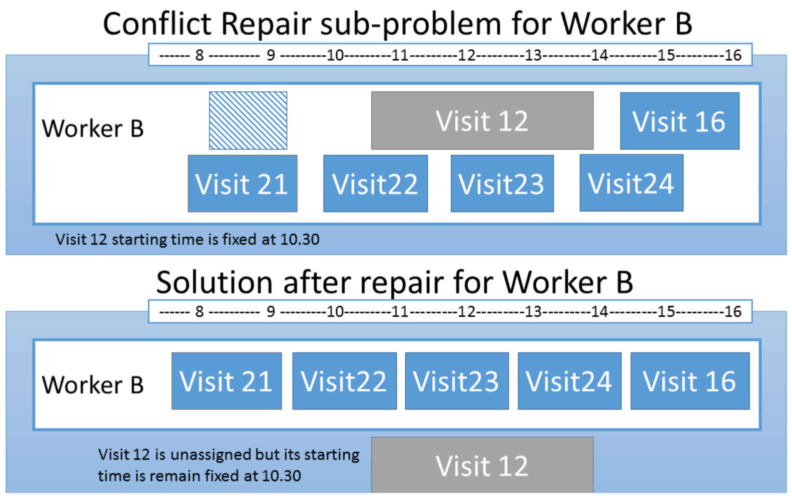
   Again, Visit 12 and Visit 13 are grouped in the same sub-problem, sub-problem 1. The solution to the sub-problem 1 assigns Visit 12 to Worker A and Visit 13 to Worker B where Visit 13 starts after Visit 12 for the duration of 2.5 hours, as shown in Sub-figure 6.3a. In the same iteration, Worker B is assigned in the solution of sub-problem 2, as illustrated in Sub-figure 6.3b. Therefore, paths assigned to Worker B must be repaired.



(a) Decomposition sub-problem 1 solution



(b) Decomposition sub-problem 2 solution



(c) Conflict Repair sub-problems

Figure 6.3: Illustration of time-dependent constraint modification example when the conflict assignments repair does not assign time-dependent activities. Sub-figure (a) shows solution from solving the decomposition sub-problem 1 where a minimum starting time differences constraint has been enforced. The blue strip pattern is a duration where Visit 12 cannot be allocated. Sub-figure (b) presents another solution from solving the decomposition sub-problem 2 where another path is assigned to the Worker B. Sub-figure (c) presents conflict repair sub-problem for the Worker B, which considers assignments from paths in solutions of sub-problem 1 and sub-problem 2, and the solution after repair for the Worker B. Assumption is that the conflicting assignments repair selects Visit 12 to be an unassigned visit. The fixed starting time at 10.30 is enforced to the Visit 12 for the next iterations.

In the conflicting assignments repair, Worker B's assignments are generated as a new sub-problem, where starting time of Visit 12 is fixed at 10.30, as shown in Sub-figure 6.3c. The MIP solver tackles this sub-problem where its solution does not assign Visit 12. This makes Visit 12 to be unassigned visit where the decision to assign this visit will be made in the next iterations. However, the starting time of Visit 12 is fixed at 10.30, so that if this visit will be assigned, the time of the assignment remains satisfactory to the minimum difference constraint.

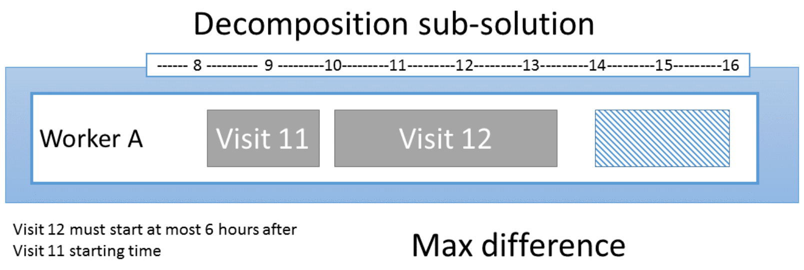
4. **Both time-dependent visits are assigned to a worker where the path requires conflicting repair and the solution after repair drops one of the dependent visits.**

   This case is applied when a worker has been used in two or more decomposition solutions where each solution has a path for that worker.

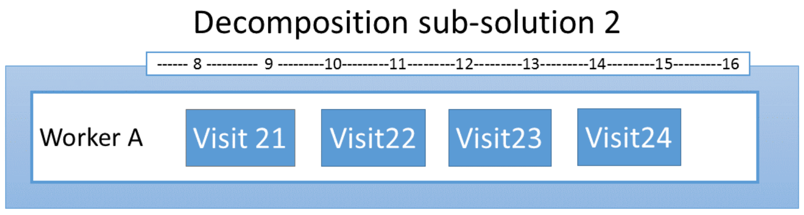
   Figure 6.4 illustrates an example of this case with a maximum difference constraint. The figure has three sub-figures: two sub-figures show two decomposition sub-problem solutions, and the other sub-figure presents a conflicting assignment repair sub-problem and its solution. For this example, Visit 11 and Visit 12 are dependent where Visit 12 must take place no later than six hours after the starting time of Visit 11. Therefore, Visit 11 and Visit 12 are grouped in the same sub-problem, sub-problem 1.

   The solution to the sub-problem 1 assigns Visit 11 and Visit 12 to Worker A where Visit 12 starts 2 hours after the starting time of Visit 11, as shown in Sub-figure 6.4a. In the same iteration, Worker A is assigned in the solution of sub-problem 2, as illustrated in Sub-figure 6.4b. Therefore, paths assigning to Worker A must be repaired.

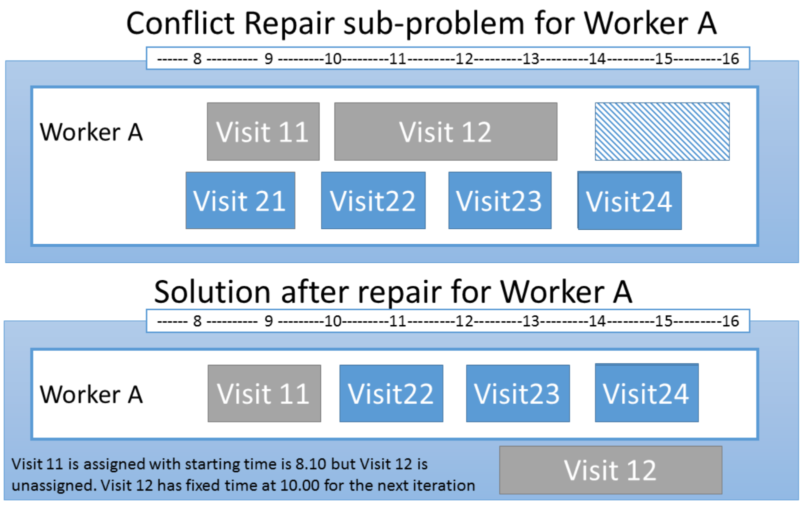
   Worker A's assignments form a new sub-problem in the conflicting assignments repair, as shown in Sub-figure 6.4c. The time-dependent modi-



(a) Decomposition sub-problem 1 solution



(b) Decomposition sub-problem 2 solution



(c) Conflict Repair sub-problems

Figure 6.4: Illustration of time-dependent constraint modification example when the conflict assignments repair does not assign one of the time-dependent activities. Sub-figure (a) shows solution from solving the decomposition sub-problem 1 where a maximum starting time differences constraint has been enforced. The blue strip pattern is a duration where Visit 12 cannot be allocated. Sub-figure (b) presents another solution from solving the decomposition sub-problem 2 where another path is assigned to the Worker A. Sub-figure (c) presents conflict repair sub-problem for the Worker A, which considers assignments from paths in solutions of sub-problem 1 and sub-problem 2, and the solution after repair for the Worker A. The Assumption is that the conflicting assignments repair selects the Visit 11 to be assigned but does not assign the Visit 12. The fixed starting time at 10.00 is enforced to the Visit 12 for the next iterations.

fication takes place by fixing Visit 11 and Visit 12 starting time at 8.10 and 10.00, respectively. The MIP solver solves this sub-problem where the Visit 11 is assigned to the Worker A at 8.10 but the Visit 12 is dropped. Therefore, Visit 12 becomes an unassigned visit where the decision to assign this visit will be made in the next iterations. However, the starting time of the Visit 12 is fixed at 10.00 so that if this visit will be finally assigned, the assignment remains satisfactory with respect to the maximum difference constraint.

From the above example, the question remaining to explain is the case when the dependent activities, such as Visit 12 in case 4, will not be assigned by any iterations. The final solution without dependent visit remains feasible to the MIP definition. This can be explained by looking back to the time-dependent activities constraints to the original problem. The constraint is presented as:

$$w_i^L y_i + \sum_{k \in K} a_i^k + s_{i,j} \leq \sum_{k \in K} a_j^k + w_j^U y_j \quad \forall i,j \in S$$

We can see that if both paired visits i,j are assigned, thus $y_i = 0$ and $y_j = 0$, then time-dependent constraint is enforced. However, the constraint also covers other cases. We can see that if

1. $y_i = 1$ and $y_j = 0$, then $w_i^L + s_{i,j} \leq \sum\limits_{k \in K} a_j^k$
2. $y_i = 0$ and $y_j = 1$, then $\sum\limits_{k \in K} a_i^k + s_{i,j} \leq w_j^U$
3. $y_i = 1$ and $y_j = 1$, then $w_i^L + s_{i,j} \leq w_j^U$

We recall that $\sum\limits_{k \in K} a_i^k = 0$ when $y_i = 1$. First, we assume that both visit i and j has been assigned by decomposition sub-problems in the first iteration and only visit i is finally applied to the final solution where the visit j is unassigned, i.e. case 2.

From constraint (6.2), $\sum_{k \in K} \sum_{i \in V^S} x_{i,j}^k + y_j = 1, \forall j \in T$; there is only one $k^* \in K$ that $x_{i,j}^{k^*} = 1$ for each visit j where $y_j = 0$. For this reason, $w_j^L \leq a_j^{k^*} \leq w_j^U$ for the worker k^*.

In addition, $x_{i,j}^k = 0$ for the other workers $k \in K$ at same visit j. From constraint (6.7) and (6.8), $a_j^k = 0$. Thus, we can simply say that $\sum_{k \in K} a_j^k = a_j^{k^*}$ where a worker k^* is assigned to visit j, ($x_{i,j}^{k^*} = 1$). Therefore, $\sum_{k \in K} a_i^k + s_{i,j} \leq \sum_{k \in K} a_j^k$. We can see that $\sum_{k \in K} a_i^k + s_{i,j} \leq w_j^U$ because $\sum_{k \in K} a_j^k \leq w_j^U$.

In the same way, the constraint $\sum_{k \in K} a_j^k + s_{j,i} \leq \sum_{k \in K} a_i^k$ is also enforced to the time-dependent pair. Since $w_j^L \leq \sum_{k \in K} a_j^k$; $w_j^L \leq \sum_{k \in K} a_i^k - s_{j,i}$.

Therefore, time windows of the visit j is enforced to the visit i such that $w_j^L + s_{j,i} \leq \sum_{k \in K} a_i^k \leq w_j^U - s_{i,j}$. The arrival time assigned in the first iteration satisfies other cases when the time-dependent activity becomes unassigned visit.

If a visit is not assigned in solutions of decomposition sub-problem in the first iteration, the visit always become an unassigned visit in the final solution. This can be shown by the BF workforce selection algorithm (see 'Workforce Selection' on page 127). The algorithm selects workers where the number of selected workers must be less than or equal to the number of visits in a subset, when the number of workers is more than the number of visits in a subset. Solving the sub-problem to optimality guarantees that all visits must be assigned to one of the workers unless the selected workers do not have any availability for the unassigned visits. Thus, this visit will be an unassigned visit to the full problem.

The number of workers in the whole problem can be less than the number of visits in a visit subset. We assume again that a visit j is not assigned by solution of decomposition sub-problem in the first iteration. Therefore, the visit will be unassigned in the final solution. For the workers who do not have conflicting paths, they will enter the final solution, thus they will not make the visit j.

The workers who have conflicting paths will enter the conflicting assignments repair where a working path will assign to each of these workers; thus, they will be added to the final solution after the conflicting assignments repair. The last case is for workers who have not been assigned by a solution for decomposition sub-problem in the first iteration. We can see that if they can make visit j, the MIP solver should assign one of the workers to make the visit because these worker have full availability. Therefore, the visit j still is unassigned.

6.3 Experiments and Results

This section describes the experiments used to compare the RDCR method to the greedy heuristic (GHI) described in Castillo-Salazar et al. [36]. This section explains the WSRP instances, overview of GHI algorithm, computational results, algorithm performance according to problem difficulties, and algorithm performance on producing acceptable solution.

6.3.1 Instance Sets of the Workforce Scheduling and Routing Problem

This study applies the RDCR method to the WSRP instances presented in [35, 36]. Those problem instances were generated by adapting several WSRP from the literature. The problem instances are categorised in four groups: Sec, Sol, HHC and Mov. The Sec group contains instances from a security guards patrolling scenario [94]. The Sol group are instances adapted from the Solomon dataset [118]. The HHC group are instances from a home health care scenario [107]. Finally, the Mov group originates from instances of the vehicle routing problem with time windows [37]. The total number of instances accumulated in these four groups is 374. The adaptations are necessary because the original problems have different features of WSRP. For example, the Sol and Mov groups do not have preferences, skill requirements, worker's proficiencies, and more importantly, the time-dependent activities requirements. Details of each instance is provided in Appendix C and summaries of adaptation made to these problem sets is described below.

Security Guards Patrolling Instances (Sec)

The original data provide a 30-day instance of the real-world security guard patrolling rounds in several locations, provided by Misir et al. [94]. The instance has visits divided into six patrol districts. The problem is to manage security guards and route them to make visits to multiple locations within a patrol district. There are 16 different skills for guards to match requirements of activities during visits. Time horizon is set to 24 hours. The problem also includes rostering where workforces are managed across the week to ensure they have enough breaks and days off.

Adaptions were made to instances in this set to transform a 30-day instance to daily problem instances. As the original data provides a month of activities, 180 instances were generated by each day and each district. Noting that the rostering constraints were also removed. All workers who are not available on a particular day were not included in a daily problem. In addition, there were changes so that some visits require two workers, with probability of 0.2. Time-dependent activities were also added to this problem.

Solomon's Instances (Sol)

The original version of this set has 56 instances, originally proposed by Solomon [118]. The original problem is a vehicle routing problem with time windows where the objective is to find the minimum number of vehicles to provide visits to every requirement. Each instance has 100 visits. Although, instances are classified into six group according to the location of visits: R100, R200, C100,

Table 6.5: Summary of Solomon's instances.

Set	R100	R200	C100	C200	RC100	RC200
# Visits	100	100	100	100	100	100
Distribution	Random	Random	Cluster	Cluster	Mixed	Mixed
Time window	Mixed	Mixed	Mixed	Mixed	Mixed	Mixed
Visit time (Mins)	10	10	90	90	10	10
Horizon (Mins)	230-240	>900	>900	>900	230-240	>900

C200, RC100, and RC200. The letter indicates type of distribution i.e. 'R' refers to random visit distribution, 'C' refers to clusters of visits, and 'RC' refers to mixed of random visit distribution and clusters of visits. Table 6.5 summarises characteristics of the original instances. The table lists instance groups by columns and characteristic by rows. From the table, time windows of all sets are mixed, which means time window of visits can be exact time or flexible. Visit times in R100, R200, RC100, and RC 200 are 10 minutes per visit and the visit times in C100 and C200 are 90 minutes per visit. Time horizons in group R100 and RC100 are between 230 and 240 minutes while the rest have the time horizon longer than 900 minutes. Note that distances between places are Euclidean.

This set requires modifications to become a WSRP with time-dependent activities constraints. First, their workers (vehicles) must be defined. Thus, Castillo-Salazar et al. [36] defined that each set has 20 workers, which is 20% of the number of visits. This proportion was also used in Bredstrom and Ronnqvist [26]. In addition, a version of 25 visits and a version of 50 visits were also created with the same proportion of workers. The instances created have also been modified by introducing a skill to both workers and visits. The approach is to use single skill with level of proficiency. A visit then requires a minimum proficiency so that only workers who have higher proficiency can make the visit. Furthermore, some visits are modified to require two workers to perform tasks, probability of 0.1 for two workers and 0.9 for one. There are four-level preferences adding to the instances: low (0.2), neutral (0.5), preferred (0.5), and most preferred (1.0). Time-dependent activities are also added to the problem. Note that worker's availability is equal to the time horizon.

Home Healthcare Instances (HHC)

This set has 11 instances extracted from real-world home healthcare scenarios. This problem is to allocate workers to make visits in multiple locations. There are four different skills distributed among the workers. There are preferences from both workers and customers. Visits are associated with priority levels in order to decide visits that should be left unassigned. Normally, the priority of unassigned visits will increase in the next scheduling day. These instances also feature time-dependent activities. There were no major changes into these instances as they have full WSRP features.

Vehicle Routing Problem with Time Window (Mov)

Instances in this set were first presented in Castro-Gutierrez et al. [37]. The instances were derived from real-world scenarios. Although, the key difference between this set and Solomon's instances is that the distances and times provided are actual travel distances based on maps, hence, they are not Euclidean and the distances and times are not symmetric. Instances are grouped into three types based on the number of visits: 50, 150 and 250 visits. Each group has five instances differed in time windows. There are 15 instances in total.

1. Available through all planning horizon;
2. 3 time windows: Morning (0-160 mins), Afternoon (160-320 mins), and Late (360-480 mins);
3. 3 short time windows: Morning (0-130 mins), Afternoon (175-305 mins), and Late (350-480 mins);

4. 3 restricted time windows: Morning (0 - 100 mins), Afternoon (190-290 mins), and Late (380-480);
5. Random time windows from the above setup.

The original data has the same structure as Solomon's instances. Therefore, the same modifications to the Solomon's instances are also applied to this set.

6.3.2 Overview of Greedy Heuristic GHI

A greedy constructive heuristic tailored for the WSRP with time-dependent activities constraints was proposed by Castillo-Salazar et al. [36]. The algorithm starts by sorting visits according to some criteria such as visit duration, maximum finish time, maximum start time, etc. Then, it selects the first unassigned visit in the list and applies an *assignment process*. For each visit $j \in T$, the *assignment process* selects all candidate workers who can undertake visit j (considering required skills and availability). If the number of candidate workers is less than the number of workers required for visit j, this visit is left unassigned. If visit j is assigned, visits $j' \in T$ that are dependent on visit j are processed. These dependent visits j' jump ahead in the *assignment process* and are themselves processed in the same way (i.e. processing other visits dependent on j'). The GHI stops when the unallocated list is empty and then returns the solution.

6.3.3 Computational Results

We applied the proposed RDCR method to the 374 instances and compared the solutions obtained to the results reported for the greedy heuristic algorithm (GHI). We mainly compare the RDCR result to the GHI because the GHI is a heuristic algorithm that can find a feasible solution for every WSRP instance. Therefore, we compare our solutions to the GHI solutions because they are the only feasible solutions that publicly available. The solutions of the other algorithms e.g. Solomon [118], Misir et al. [94], Rasmussen et al. [107], and Castro-Gutierrez et al. [37]; are not used in this comparison because these algorithms were designed to tackle the instance without time-dependent activities constraints. Thus, their solution may not always feasible in the adapted instances.

We also compare the RDCR and GHI solutions to the solution obtained from the MIP solver when the solver solution is available. The time limit for the MIP solver is set at two hours. However, the solutions obtained from the MIP solver are not used in the main comparison because only a selected number of optimal solution can be found. Alternatively, we mainly represent the solution quality by comparing the solution to the best known solution, defined as *relative gap*. The best known solution is the best solution amongst the three solution approaches: the MIP solver, the GHI, and the RDCR. The relative gap formula is written as:

$$Gap = |z - z^b| / |z^b|$$

where z represents an objective value of a solution and z^b is an objective value of the best known solution. If a solution is the best known solution, then its $Gap = 0$.

First, the related-samples Wilcoxon Signed Rank Test [63] was applied to examine the differences between the two algorithms, GHI and RDCR. The significant level of the statistical test was set at $\alpha = .05$. Results of this statistical test using SPSS are shown in Table 6.6. The table shows that RDCR produced 201 better solutions out of the 374 instances. However, there was no statistical significant difference in the solution quality between the two methods.

Figure 6.5 and Figure 6.6 compare the number of best solutions found by each of the two methods and the average relative gap to the best known solutions, results are grouped by dataset. Regarding the number of best solutions,

Table 6.6: Statistical result from Related-Samples Wilcoxon Signed Rank Test provided by SPSS

Total N		374
# of (RDCR $<$ GHI),	RDCR is better than RDCR	201
# of (RDCR $>$ GHI),	GHI is better than GHI	173
Test Statistic		38,257
Standard Error		2,092
Standardized Test Statistic		1.527
Asymp. Sig. (2-sided test)		.127

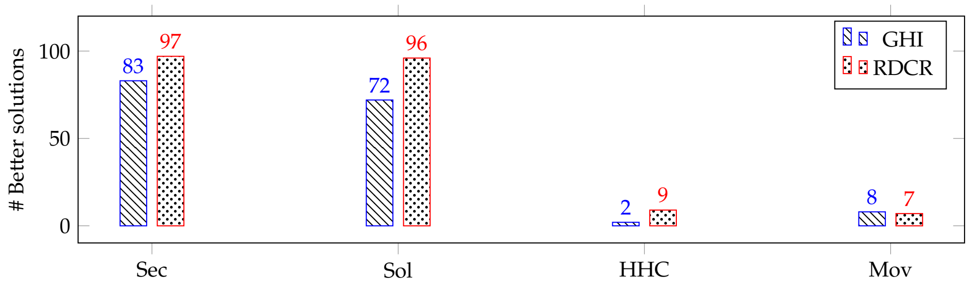


Figure 6.5: Number of best solutions obtained by GHI and RDCR for each dataset.

RDCR produced better results than GHI on three datasets: Sec, Sol and HHC. Results also show that RDCR found lower average relative gap on three datasets: Sec, Sol and HHC. However, GHI found lower relative gap on the Mov dataset.

On datasets Sec and Sol, RDCR found slightly better results than GHI as shown by the number of best solutions and the average relative gap. In dataset Sec, RDCR and GHI gave 11% and 18% of average relative gap respectively. This indicates that both algorithms provide good solution quality compared to the best known solution. On the other hand, RDCR and GHI produced 1,216%

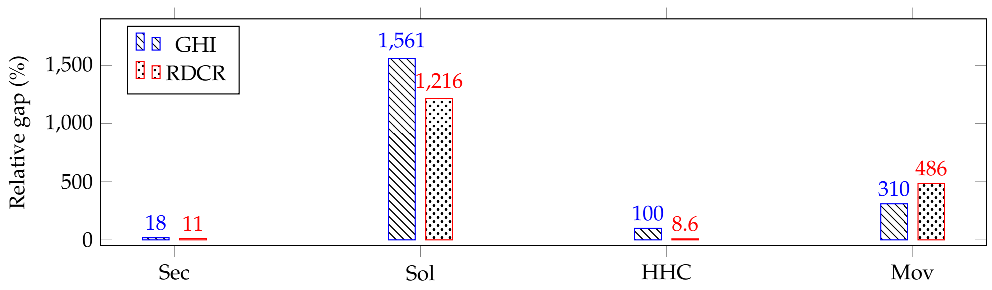


Figure 6.6: Average relative gap (relative to the best known solution) obtained by GHI and RDCR. The lower the bar the better, i.e. the closer to the average best known solution.

and 1,561% respectively for the average relative gap to the best known solution in dataset Sol. This implies that both algorithms failed to find solutions that are of competitive quality to the best known solution, but both algorithms are competitive with each other. We can see that instances in this Sol dataset are particularly difficult as neither the GHI heuristic nor the RDCR decomposition technique could produce solutions of similar quality to the best known solution.

On dataset HHC, the average relative gap of RDCR is much lower than the average gap of GHI. The results show that RDCR has 8.67% relative gap while GHI has 100.4%. For the HHC instances, RDCR found the best known solution for 9 instances and GHI found the best known solution for the other 2 instances. For these two instances, average relative gap of RDCR is 47%. However, in the 9 best solutions of RDCR, average gap of GHI is 109%. A closer look at the Sol dataset showed that these instances have priority levels defined for the visits. It turns out that GHI does not have sorting parameters to support such priority for visits because the algorithm sorting parameters focuses on the time and duration of visits. On the other hand, RDCR implemented priority for visits within the MIP model. This could be the reason that explains the better results obtained by RDCR on this particular dataset.

On dataset Mov, GHI gives better performance. GHI delivers 8 better solutions (7 best known) from 15 instances while RDCR gives 7 better solutions (4 best known). The average relative gap of GHI is 310% which is less than the 486% relative gap provided by RDCR. There are 5 instances which best known solution is given by the mathematical solver. The average relative gaps to the best known by GHI and RDCR are 315% and 36% respectively. We found that the decomposition method does not show good performance on this particular Mov dataset, especially on instances with more than 150 visits. The main reason is that the solver cannot find optimal solutions to the sub-problems within the given time limit. Therefore, the size of sub-problems in these Mov instances

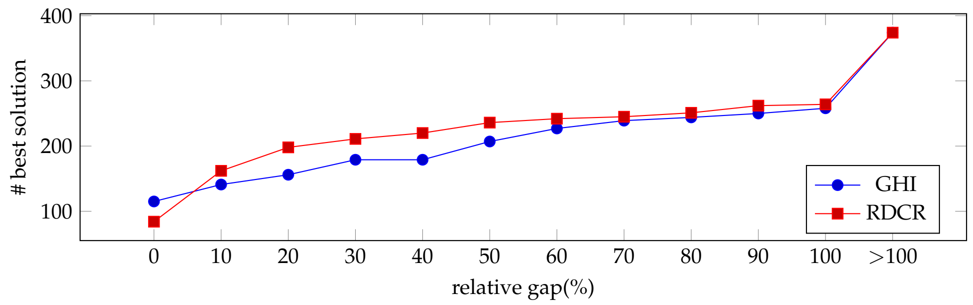


Figure 6.7: Cumulative distribution on relative gap by RDCR and GHI.

should be decreased to allow for the sub-problems to be solved to optimality.

Figure 6.7 shows the cumulative distribution of RDCR and GHI solution over the relative gap. It shows the number of solutions which have a relative gap to the best known less than the corresponding value in the X-axis. Note that 0% relative gap refers to the best known solution. GHI provides 115 best known solutions which is better than RDCR which provides 84 best solutions. This is represented by the two leftmost points in the figure. However, from the value of 10% relative gap onwards, RDCR delivers more solutions which reach relative gap percentages than GHI. In general, apart from the overall number of best known solutions, RDCR provides higher number (or equal) of solutions than GHI for different values of relative gap. For example, if we set the solution acceptance rate at 50% relative gap, RDCR produces 236 solutions of this quality while GHI produces 207. RDCR delivers overall more solutions with acceptance rate up to 100% gap to the best known.

Figure 6.8 shows the distribution of computational time spent by the proposed RDCR method when solving the WSRP instances considered here. These results show that RDCR spends more computational time on most of the HHC instances with an overall average time spent on each instance of 2.4 minutes. Note that the highest computational time observed in these experiments is less than 74 minutes. Therefore, GHI is clearly superior to RDCR in terms of computational time.

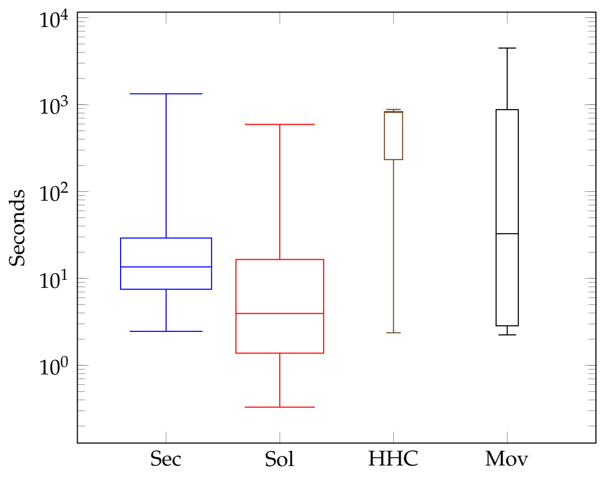


Figure 6.8: Box and Whisker plots showing the distribution of computational time in seconds spent by RCDR for each group of instances. The wider the box the larger the number of instances in the group. The orange straight line presents the upper bound of the computational time spent by GHI fixed to 1 second. The Y-axis is in logarithmic scale.

6.3.4 Performance According to Problem Difficulty

This part seeks to better understand the performance of the two algorithms GHI and RDCR. For this, a more detailed analysis is conducted of the instances in which each of the algorithms performs better than the other one. Then, the problem features are analysed in detail in order to unveil any conditions under which each of the algorithms appears to performs particularly well.

Table 6.7 presents the main characteristics of the problem instances in three groups. *Set All* has all of the 374 instances. *Set GHI* has all problem instances in which GHI produced better solutions than RDCR. *Set RDCR* has all problem instances in which RDCR produced better solutions than GHI. The table presents five main columns. The first column shows type of characteristics to be investigated. The second main column shows descriptive statistic of the data. This column shows two values: median and interquartile range (IQR) which presented in two sub-columns. The third and forth main columns present mean ranks of each characteristic in set GHI and set RDCR, respectively. The mean rank is calculated using Mann-Whitney U test. The last column presents calculated

Table 6.7: Summary of the problem features for different groups of problem instances. The *Set All* includes all instances. The *Set GHI* includes the instances in which GHI produces better solutions than RDCR. The *Set RDCR* includes the instances in which RDCR produces better solutions than GHI.

Group	*Set All*		*Set GHI*	*Set RDCR*	Sig.
# Instances	374		165	209	(2-tailled)
	Median	IQR	Mean Rank	Mean Rank	
Problem Size					
#Worker	16.00	21.00	242.35	144.20	<.001
#Visit	69.00	73.25	246.18	141.17	<.001
Characteristics on Visits and Workers					
VisitDur	90.00	390.11	212.18	168.02	<.001
#TimeDep	12.00	14.00	234.05	150.75	<.001
Worker/Visit	1.18	0.06	176.25	196.38	.072
WorkerHours	24.00	4.00	194.52	181.96	.239
VisitWindow	426.01	334.97	200.88	176.94	.033
Horizon	1440.00	480.00	194.52	181.96	.239

statistical significant value provided by Mann-Whitney U test. We set significant level at $\alpha = .05$. The Mann-Whitney U test is used here because our data does not have normal distribution. We investigate 8 problem characteristics: the number of workers (#Worker), the number of visits (#Visit), visit duration (VisitDur), the number of time-dependent activities (#TimeDep), worker-visit ratio (Worker/Visit), worker available hours (WorkerHours), average visit time window (VisitWindow), and planning horizon (Horizon). .

It seems obvious to relate the difficulty of a particular problem instance to its size, which can be measured by the number of workers and the number of visits. It could also be assumed that the length of the planning horizon might have some influence on the difficulty of the problem in hand, although perhaps to a lesser extent than the number of workers and visits. However, the analysis presented here seeks to identify other problem characteristics that might have an effect on the difficulty of the instances when tackled by each of the algorithms RDCR and GHI. For example, it can be argued that having visits with longer duration or large number of time-dependent activities could make the problem instance more difficult to solve because of the higher likelihood of time conflicts arising. In contrast, the difficulty could decrease for a problem instance that has higher worker to visit ratio (i.e. more workers to choose from), longer worker working hours or wider visit time windows (i.e. more flexibility for the assignment of visits).

Considering the above, it seems from Table 6.7 that instances in *Set RDCR* are less difficult than those in *Set GHI*. In respect of the problem size, instances in *Set RDCR* are on average smaller than those in *Set GHI*, on the number of workers (#Worker) and also the number of visits (#Visit). In addition, instances in *Set RDCR* have shorter visit duration (VisitDur), lower number of time-dependent activities (#TimeDep), and shorter visit time window (VisitWindow) than instances in *Set GHI*. The differences between the two sets in respect of the remaining three problem characteristics: worker-visit ratio (Worker/Visit), worker available hours (WorkerHours), and planning horizon (Horizon) were found to be not statistically significant.

Then, from the above analysis, it can be argued that the RDCR approach performs better than GHI on instances of lower difficulty level. However, establishing the boundary between lower and higher difficulty is not so clear given the overlap in values for the 8 problem characteristics between *Set RDCR* and *Set GHI*. Hence, the proposal here is to recommend the use of RDCR for instances with less than 16 workers and less than 69 visits (the median of all 374 instances), and the use of GHI otherwise. This recommendation can be used as a first step for choosing between RDCR and GHI.

6.3.5 Performance on Producing Acceptable Solutions

The previous subsection sought to identify a boundary in problem difficulty between those instances in which each of the methods RDCR and GHI performs better than the other one. This subsection seeks to identify instances for

Table 6.8: Summary of the problem features for different groups of problem instances. The group *Accept Heur* includes instances for which an acceptable solution was found by at least one of the two heuristic algorithms RDCR and GHI. The group *Reject Heur* includes instances for which none of RDCR or GHI delivers an acceptable solution.

	Reject Heur	*Accept Heur*	Sig.
# Instances in Group	79	295	(2-tailed)
	Mean Rank	Mean Rank	
Problem Size			
#Worker	100.02	210.93	<.001
#Visit	103.69	209.94	< .001
Characteristics on Visits and Workers			
VisitDur	101.94	210.41	<.001
#TimeDep	100.09	210.91	<.001
Worker/Visit	168.59	192.56	.079
WorkerHour	120.55	205.43	<.001
VisitWindow	150.96	197.29	<.001
Horizon	120.55	205.43	<.001

which both algorithms can deliver acceptable solutions. For this, a solution that has a relative gap of at most 100% with respect to the best known solution is considered acceptable, otherwise it is labelled unacceptable.

The first part of the analysis splits the problem instances into two groups. The group *Accept Heur* has instances for which an acceptable solution was found by at least one of the two heuristic algorithms RDCR and GHI. The group *Reject Heur* has instances for which neither of RDCR or GHI delivers an acceptable solution. Basically, this analysis seeks to identify a boundary in problem difficulty for which the methods RDCR and GHI can perform better than an exact solver. Table 6.8 shows the problem characteristics for the two groups *Accept Heur* and *Reject Heur*. Each row shows mean rank calculated by Mann-Whitney U test for each of 8 problem characteristics. The column "Sig (2-tailed)" shows calculated statistical value for each characteristic using Mann-Whitney U test. We set significant level $\alpha = .05$.

The results in Table 6.8 show that there are significant differences between

Table 6.9: Summary of the problem features for different groups of problem instances. The group *Accept GHI* includes instances for which an acceptable solution was found by algorithm GHI, otherwise the instance is included in group *Reject GHI*.

	Reject GHI	*Accept GHI*	Sig.
# Instances in Group	37	258	(2-tailed)
	Mean Rank	Mean Rank	
Problem Size			
#Worker	66.19	159.73	<.001
#Visit	71.66	158.95	<.001
Characteristics on Visits and Workers			
VisitDur	65.81	159.79	<.001
#TimeDep	71.92	158.91	<.001
Worker/Visit	132.39	150.24	.233
WorkerHour	117.91	152.32	.010
VisitWindow	147.20	148.11	.952
Horizon	117.91	152.32	.010

the groups *Accept Heur* and *Reject Heur* on seven problem characteristics. That is, the group *Accept Heur* shows higher mean ranks than the group *Reject Heur* for the number of workers (#Worker), the number of visits (#Visit), visit duration (VisitDur), the number of time-dependent activities (#TimeDep), worker-visit ratio (Worker/Visit), worker available hours (WorkerHours), and planning horizon (Horizon). These results indicate that GHI and RDCR do not provide acceptable solutions on the smaller instances as the lower rank means less value on that characteristic. However, heuristic algorithms do well on the larger instances. This is because the exact solver performs very well on the smaller instances but not so well when the problem size grows.

The second part of the analysis splits the 295 problem instances from the group *Accept Heur* into groups according to whether the particular method GHI or RDCR produces acceptable solutions or not. As before, a solution that has a relative gap of at most 100% with respect to the best known solution is considered acceptable, otherwise it is labelled unacceptable. Table 6.9 shows the split for method GHI into groups *Accept GHI* with 258 instances and *Reject GHI*

Table 6.10: Summary of the problem features for different groups of problem instances. The group *Accept RDCR* includes instances for which an acceptable solution was found by algorithm RDCR, otherwise the instance is included in group *Reject RDCR*.

	Reject RDCR	*Accept RDCR*	Sig.
# Instances in Group	31	264	(2-tailed)
	Mean Rank	Mean Rank	
Problem Size			
#Worker	177.97	144.48	.038
#Visit	169.61	145.46	.135
Characteristics on Visits and Workers			
VisitDur	64.68	157.78	<.001
#TimeDep	149.63	147.81	.910
Worker/Visit	60.65	158.26	<.001
WorkerHour	101.77	153.43	<.001
VisitWindow	127.05	150.46	.148
Horizon	101.77	153.43	<.001

with 37 instances. There are significant differences between the two groups on six characteristics: the number of workers (#Worker), the number of visits (#Visit), visit duration (VisitDur), the number of time-dependent activities (#TimeDep), worker available hours (WorkerHours), and time horizon (Horizon) with higher ranks for the group *Accept GHI*. These results confirm that GHI provides acceptable solutions on the larger instances but it struggles to produce acceptable solutions for some smaller instances.

Table 6.10 shows the split for method RDCR into groups *Accept RDCR* with 264 instances and *Reject RDCR* with 31 instances. There are significant differences between the two groups on five characteristics: the number of workers (#Worker), visit duration (VisitDur), worker-visit ratio (Worker/Visit), worker available hour (WorkerHour), and time horizon (Horizon). The size of instances in group *Accept RDCR* seems smaller than in group *Reject RDCR* as given by mean ranks of #Worker and #Visit, although only for #Worker that the difference is significant. Instances in the group *Reject RDCR* have shorter visit duration and lower worker-visit ratio. A problem instance could become more

Table 6.11: Summary of recommended approaches to tackle WSRP based on problem size and number of instances in each size class.

Algorithm	Exact Method	RDCR	Heuristic	GHI
#Instance	79	37	227	31
Problem Size	Very Small	Small	Medium	Large
Average #Worker	9.91	15.97	23.42 - 27.29	49.74
Average #Visit	49.39	54.86	95.20 - 103.43	155.6

difficult to solve if there are less workers to be assigned to visits. These results confirm that the performance of RDCR on providing acceptable solutions suffers as the size of the problem grows.

From the above analysis on producing acceptable solutions, some recommendations can be drawn in respect of what type of approach to use according to the problem size. Table 6.11 shows the type of approach recommended according to the problem size and number of instances in each size class. The first row of the table shows the suggested algorithm for each size class, Heuristic refers to either GHI or RDCR. For each size class, the table shows the number of instances (#Instance), the problem size label, the average number of workers (Average #Worker) and the average number of visits (Average #Visit). It is suggested that to use the exact method to solve very small instances, to use RDCR to solve small and medium instances and to use GHI to solve medium and large instances. The problem size class with the largest number of instances is the medium class for which the two heuristic algorithms, GHI and RDCR, find acceptable solutions. These recommendations in Table 6.11 were drawn from looking at the reject groups in Tables 6.8 to 6.10. Both GHI and RDCR do not perform well when solving small instances, given that group *Reject Heur* in Table 6.8 has the smallest average problem size. RDCR should be used for instances larger than those in group *Reject Heur*, Table 6.9 shows that the *Reject GHI* group has average problem size larger than the *Reject Heur* group and smaller than the *Reject RDCR* group. GHI tends to be effective in the largest instance group, it can be seen from Table 6.10 that the *Reject RDCR* group has the largest average problem size compared to the *Reject GHI* group and *Reject Heur*. However, both RDCR and GHI have similar performance as their acceptable solutions are similar in number.

6.4 Conclusion

This chapter described the modification to the Repeated Decomposition and Conflict Repair in order to solve instances of the workforce scheduling and routing problem (WSRP) with time-dependent activities constraints. We use heuristic partition and selection to split a problem into sub-problems. The sub-problem is individually solved by the MIP solver. Within a sub-problem solution, all paths get satisfaction from all constraints. Although, paths may conflict with other paths provided by other sub-problems, this can be fixed by the conflicting assignments repair process. However, the conflicting assignments repair requires a modification to support time-dependent activities constraints since the conflicting assignments repair may rearrange the assigned times. Thus, the modification maintains the layout of the time-dependent activities by fixing the time assigned by the solution of decomposition sub-problems of the time-dependent visits. Then, the conflicting assignments repair rearranges the assignments which do not have time-dependent visits to find a valid path. As a result, paths generated by the conflicting assignments repair satisfy all constraints of the full model where the paths can be used in the final solution.

The proposed RDCR approach is applied to solve four WSRP scenarios which provide a total of 374 instances. The experimental results showed that RDCR was able to find solutions which are better than solutions of the GHI heuristic for 209 instances. However, the statistical test showed that the average quality of RDCR solutions does not differ from the average quality of GHI solutions significantly. The analysis by the group of instances showed that the RDCR had a higher number of better solutions than GHI on three datasets.

The computational time required to solve a problem instance using the RDCR ranges from less than a second to 74 minutes. The average computational time was under three minutes. However, GHI solved an instance with less than a second. This has been shown clearly that the GHI has the edge over the RDCR on computational time. However, the average computational time at three minutes is acceptable because the acceptable computational time for the WSRP instances were set at two hours by Castillo-Salazar et al. [36]. Overall, the RDCR with time-dependent modification was able to effectively solve WSRP instances with time-dependent activities constraints. The method found competitive feasible solutions to every instance and within reasonable computational time.

Our future work is towards improving the computational time of the proposed RDCR approach. Such improvement might be achieved by applying different methods to partition the set of visits or by using more effective workforce selection rules. Also, determining the right sub-problem size could be interesting as it could help to balance solution quality and time spent on computation.

Chapter 7

Model Reformulation of the Home Healthcare Problem

The previous chapters explained decomposition methods splitting the main problem instance into smaller sub-problems. This chapter, on the other hand, introduces a compact mathematical formulation to solve each of the 42 HHC instances without decomposing them.

The content of this chapter has not yet been published. The manuscript is being prepared which is aimed to submit to an operations research journal by the end of 2016.

This chapter tackles the home healthcare problem (HHC) by reformulating the full model presented in Chapter 2 into a compact model. The compact model to solve this problem is in a form of an assignment problem. We acknowledge that the full model supports all WSRP features, including some which might not fully be required by some HHC instances, such as time windows, soft constraint violation costs, and exact routing costs. The solution to the full model provides all details of an assignment such as assigning cost, travelling cost, constraint violation cost, worker's route, etc. However, we found evidence from previous chapters that the HHC instances represented by the full model cannot be solved to optimality with reasonable computing resources, e.g. physical memory, computational time, etc. Experiments in this chapter compare the compact model and the decomposition methods to solve problem instances presented in Section 2.5.

7.1 Model Reformulation in the Literature

Reformulations are processes of changing mathematical formulations into another equivalent form, which is usually easier to solve [86, 87]. Reformulations can be done by model elements such as lifting (adding additional variables to the model), restriction (replacing variables with parameters), projection (removing variables), converting equations to inequalities, etc. The reformulations can change the model types such as changing non-convex nonlinear programming into bilinear terms with linear constraints [84, 95, 115–117] known as linearisation. Note that problems in nonlinear form usually have complications to find an optimal solution; thus a linear model is generally preferred. Below, we present examples of using mathematical reformulation in the literature.

Generally, the literature proposes automated reformulation process which identifies structured constraints and reformulates them [52, 109, 110]. The aim of automated reformulation is to detect and change formulations which often make a solver time-consuming, such as symmetric optima, i.e. no branch-and-bound nodes can be pruned [85]. The modification is then made to break the problem symmetry which increases the branch-and-bound convergence speed. Alternatively, a reformulation can also use redundant information to generate cutting planes, which then helps to reduce computational time [3]. This approach can be done automatically by applying a cutting plane algorithm [69, 73].

A problem specific reformulation is also widely used, for example projecting a directed Steiner tree problem into the binary arc variable [125]. This approach requires deeper understanding of the problem but could deliver tight formulations in a few lines. From Vanderbeck and Wolsey [125], the directed Steiner tree problem is defined in a graph $G = (V, E)$ with its edge cost $c_{i,j} \in \mathbb{R}$. A root node $d \in V$ and a set of terminals $T \subseteq V \setminus \{d\}$ are defined. The problem is to find a minimum cost subgraph containing a directed path from d to each node in T, which the formulations can be written in the form:

$$\text{Min} \sum_{i,j \in V} c_{i,j} x_{i,j} \tag{7.1}$$

$$-\sum_{j \in V^S} w_{d,j} = -|T| \tag{7.2}$$

$$-\sum_{j \in V^S} w_{i,j} + \sum_{j \in V^N} w_{j,i} = 0 \quad , \forall i \in V \setminus (T \cup \{d\}) \tag{7.3}$$

$$-\sum_{j \in V^S} w_{i,j} + \sum_{j \in V^N} w_{j,i} = 1 \quad , \forall i \in T \tag{7.4}$$

$$w_{i,j} \leq |T| x_{i,j} \quad , \forall i, j \in V \tag{7.5}$$

$$w \in \mathbb{R}, x \in \{0, 1\}. \tag{7.6}$$

The direct implementation is to construct a subgraph which requires $|T|$ units to flow out from d and one unit to flow into every node in T. Hence, positive variable $w_{i,j}$ represents flows between two nodes. A node in T consumes the flow by 1 to mark a visit (7.4) while the flows are balanced in the other nodes (7.3). Finally, only edges which have been used in the graph are marked as used (7.5). Any used edges will add cost to the objective function.

The problem can be reformulated as represented by the following system:

$$\text{Min} \sum_{i,j \in V} c_{i,j} x_{i,j} \tag{7.7}$$

$$\sum_{(i,j) \in g(U)} x_{i,j} \geq 1, \forall U \subseteq V \text{ such that } d \in U \text{ and } T \setminus U \neq \emptyset \tag{7.8}$$

$$0 \leq x \leq 1 \tag{7.9}$$

where $g(U)$ is a set of edges with exactly one endpoint in U. This reformulated model has $2^{|V|-1}$ constraints which is the number of all possible subset U [65]. One may assume that U is a set of nodes that has formed a Steiner tree. Thus, constraint (7.8) can be interpreted that at least one edge is required to connect the tree U to all other nodes outside the tree. Vanderbeck and Wolsey [125] explained that the reformulated model is exactly Benders' separation problem where the optimal solution of linear programming relaxation of this problem is a solution of the original problem.

The example presented above shows that the reformulation approach may increase the number of constraints as long as the reformulated problem is easier to solve. For our approach in this chapter, we use reformulation to reduce the number of constraints because the full HHC model proposed in Chapter 2 is too large to be solved when tackles large real-world instances. The rest of this chapter presents a compact model which has only a few constraint types to define the same full HHC problem. In addition, the number of constraints for the compact model is less than the full model to reduce the requirement of computational memory.

The compact model is designed based on specific problem characteristics. The redesigned model is expected to be small enough so that the optimal solution could be found. However, this implementation might cause compatibility restrictions to solve the general WSRP because the compact model is specifically designed for the HHC instances presented in Chapter 2. A solution given by solving the compact model requires a conversion to the solution format used by the full model for measuring the solution quality. Next, this chapter presents problem characteristics which becomes important for considering the compact model.

7.2 Compact Mixed Integer Programming Model for the Home Healthcare Problem

We propose a compact MIP model implemented specifically for the HHC problem. The full model has shown flexibility that can tackle the 42 HHC instances and the WSRP with time-dependent activities constraints by applying a few adaptations.

However, a closer look at the 42 HHC instances reveals that these instances have fixed visiting time while the full model presented in Chapter 2 supports a full flexibility by implementing time window constraints. That is, we can provide time window as $w_j^L = w_j^U$ for the fixed visiting time case. To support time assignment flexibility, other constraints such as travel time feasibility constraints and workforce time availability constraints are also required. The compact model does not explicitly have these three constraints. Instead, it generates compressed data, a single value to represent multiple data in the full model. The compressed data has four components: a conflict matrix, a workforce-visit compatibility matrix, a cost matrix, and a working hour limit vector. Next, we explain the compressed data, and then the compact MIP formulations.

Table 7.1: Summary of dimensions and value types of four data components for the compact model.

Component	Dimension	Data Type
Conflict Matrix Q	$\|T\| \times \|T\|$	Binary Value
Workforce-Visit Compatibility Matrix B	$\|K\| \times \|T\|$	Binary Value
Cost Matrix C	$\|K\| \times \|T\|$	Positive Real Value
Working Hour Limit Vector	$\|K\|$	Positive Real Value

7.2.1 Compressed Data

To reduce the size of the problem, we must first compress data which has common structure into matrices. The compact MIP model requires only four components: a conflict matrix, a workforce-visit compatibility matrix, a cost matrix, and a working hour limits vector. From these components, only the working hour limits vector is represented in the same way as in the full model.

The four components have different dimensions. Table 7.1 summarises the dimensions of each component and its data type. The full detail of each component is listed below.

Conflict Matrix

A conflict represents visits where visiting durations are overlapped. A conflict matrix $Q = (q_{i,j})$ is a binary matrix where its dimension is $|T| \times |T|$ where T is a set of visits. For each $q_{i,j}$ presents a time conflict between visit i and visit j. If $q_{i,j} = 1$ means visit i has a time conflict with visit j, $q_{i,j} = 0$ otherwise. Time conflicted visits cannot be made by the same worker.

The conflict matrix is built based on fixed arrival time. Thus, this data is calculated from every two visits to check that they could be overlapped by their working duration and travel times between the two visits. This is to guarantee that a worker k can make both visit i and visit j when $q_{i,j} = 0$.

For any two visits i, j, they are time conflicted ($q_{i,j} = 1$) if

$$w_i + \delta_i + t_{i,j} \geq w_j \quad \text{and} \quad w_j + \delta_j + t_{j,i} \geq w_i$$

where w_i, w_j are fixed arrival time of visit i and j; δ_i, δ_j are duration of visit i and visit j; and $t_{i,j}$ is travel time from visit i to visit j and $t_{j,i}$ is travel time from visit j to visit i respectively.

Workforce - Visit Compatibility Matrix

A Workforce-visit compatibility determines hard conditions of assigning a worker k to make visit j. A workforce-visit compatibility matrix $B = (b_j^k)$ is a $|K| \times |T|$ binary matrix generated by testing compatibility between all possible workers and visits. If a worker k is compatible to make a visit j, the value $b_j^k = 1$ and the value $b_j^k = 0$ when a worker k is prevented to make a visit j. Compatibility involves two requirements: minimum skill requirements and workforce contracts. Hence, a worker k is compatible to make a visit j if

1. worker k has qualified skills to all minimum skill requirements, and
2. worker k holds at least one contract allowing to make the visit j.

Cost Matrix

The cost matrix is a matrix containing the sum of costs incurred for a worker making a visit. The dimension of the matrix $C = (c_j^k)$ is $|K| \times |T|$. Each cost value c_j^k of matrix presents a cost to assign a worker k to make a visit j. The cost value sums up all weighted assigning costs presented in the objective function of the full model into a single value.

The fundamental of the cost matrix is the objective function (2.14) of the full model. The objective function (2.14) has four objective tiers, $\lambda_1, \ldots, \lambda_4$. For convenience, the objective function of the full model is repeated here:

$$\text{Min} \sum_{k\in K}\sum_{i\in V^S}\sum_{j\in V^N} \lambda_1 \left(d_{i,j} + p_j^k\right) x_{i,j}^k + \sum_{k\in K}\sum_{i\in V^S}\sum_{j\in V^N} \lambda_2 \rho_j^k x_{i,j}^k + \sum_{j\in T}(\lambda_3(\omega_j + \psi_j) + \lambda_4 y_j) \quad (7.10)$$

The cost matrix requires all cost value to be filled. This includes a cost of assigning a worker who does not qualify to make visit j. However, the assignment of worker k to make visit j will not be made if the cost c_j^k is larger than λ_4. Thus, we introduce λ_5 as a cost when the worker k cannot make the visit j where $\lambda_5 \gg \lambda_4$.

Since the cost matrix is calculated based on the assignment is made, thus, $y_j = 0$. Therefore, we estimate the cost of assigning worker k to visit j by:

$$c_j^k = \lambda_1(p_j^k + d_{v^k,j}) + \lambda_2\rho_j^k + \lambda_3(\omega_j^k + (1-\gamma_j^k)) + \lambda_5(1-r_j^k) \quad (7.11)$$

where

- p_j^k is a monetary cost when assigning a worker k to make a visit j,
- $d_{v^k,j}$ is an estimated distance by using a distance between a worker's starting location v^k and a visit j.
- ρ_j^k is a preference penalty cost when assigning a worker k to make a visit j,
- ω_j^k is a time availability violation parameter when assigning worker k to make visit j,
- γ_j^k is a region availability violation parameter when assigning worker k to make visit j, and
- r_j^k is a workforce visit compatibility between worker k and visit j.

As the monetary cost p_j^k is defined by workforce-task, it is the same value used in full model. The monetary cost is then multiplied by the weight λ_1.

The second part of monetary cost is the distance between a visit $j \in T$ and the worker starting location $v^k \in K$. This part of the cost is to estimate that the distance of a visit assigned to a worker has been considered in the cost matrix.

A preference penalty parameter ρ_j^k is the same preferences defined by the full model. The value is then multiplied by weight λ_2.

Soft constraint violations are pre-calculated for all combinations of workforce and tasks. There are two soft constraint violations: the time availability violation ω_j^k and the region availability violation γ_j^k. Soft constraint violations are pre-calculated by assuming a worker k makes a visit j. The algorithm then checks if the worker has time availability to make the visit ($\alpha_L^k \leq w_j$ and $w_j + \delta_j \leq \alpha_U^k$) in which case the $\omega_j^k = 0$; otherwise $\omega_j^k = 1$. A similar practice applies to the region availability violation γ_j^k where the worker availability parameter $\gamma_j^k = 1$ when a worker k is available in the region of the visit j, and $\gamma_j^k = 0$ otherwise. Both soft constraint violations are multiplied to the objective weight λ_3.

The cost also includes hard constraint violation which is a value of $1 - r_j^k$ where r_j^k is the workforce-visit compatibility value explained above. The hard constraint violation is then multiplied by weight λ_5 which results in a hard constraint penalty cost which is larger than the cost of unassigned visit λ_4. This is to guarantee than the MIP solver will select a visit to be unassigned rather than violating hard constraints.

The cost matrix plays a major role in this transformation because it simplifies features of the full model into a single matrix. The cost matrix does not reflect the actual cost of the problem because it does not include travelling costs between places. An exact travelling cost cannot be applied here according to the reduced matrix dimension.

Table 7.2: Notation used in HHC compact model.

Sets	
K	as a set of workers.
T	as a set of visits.
$\hat{T}(k)$	as special visit sets that the worker k can take, i.e. $b_j^k = 1$.
C	as a cost matrix where c_j^k is a cost of assigning worker k to make visit j
Q	as a conflicting visit matrix where $q_{i,j} = 1$ is a value indicating that visit i and visit j cannot be made by the same worker.
B	as a workforce-visit compatibility matrix where $b_j^k = 1$ is a value indicating that a worker k can make a visit j.
Parameters	
h	as a worker hour limits vector where h^k is the hour limit for a worker k.
λ_4	is a cost charged when a visit is left unassigned.
r_j	is a workforce requirement for visit j.
Variables	
x_j^k	is a binary variable indicating assignment status between worker $k \in K$ and visit $j \in T$.
y_j	is an integer variable indicates a visit $j \in T$ has been left unassigned when $y_j > 0$.

Working Hour Limits Vector

The working hour limits vector for the compact model has the same structure with the full model. For each worker $k \in K$ has limited working hours h^k which can be written as $h = (h^k), \forall k \in K$. The value of the working hour limit is taken directly from the full model.

7.2.2 Mathematical Formulations for the Compact Model

We propose a compact model to solve the WSRP. We first investigate other models in the literature which should be compatible to our approach. The similar approaches are a generalised assignment problem and a resource scheduling problem. We summarise notations used in this section in Table 7.2.

The compact model can also be seen as a generalised assignment problem

[41]. The problem is to find the cheapest way to assign $|T|$ visits to $|K|$ workers where $|T| \geq |K|$. The problem requires that a visit must be assigned to one worker as long as the workload (δ_j) is not exceeding worker's limit (h^k). Our compact model, however, has additional constraints which allow a worker to take jobs as long as jobs are not conflicting and a worker is compatible to make visits. The generalised assignment formulation can be written by:

$$\text{Minimise} \sum_{k \in K} \sum_{j \in T} c_j^k x_j^k \tag{7.12}$$

subject to

$$\sum_{k \in K} x_j^k = 1 \qquad \forall j \in V^N \tag{7.13}$$

$$\sum_{j \in T} \delta_j x_j^k \leq h^k \qquad \forall k \in K \tag{7.14}$$

$$x_j^k \quad \textit{binary} \qquad \forall j \in T, \forall k \in K \tag{7.15}$$

The compact model is also similar to a resource scheduling problem which is to find a solution to operate $|T|$ visits with $|K|$ workers [71]. Each worker can handle at most one job at a time and a job must be executed by one worker at a time. A slight difference between our compact model and the classical resource scheduling is that our problem allows a visit to be made by multiple workers. In terms of formulations, the classical resource scheduling problem is presented as a transportation model [21].

$$\text{Minimise} \sum_{k \in K} \sum_{j \in T} c_j^k x_j^k \tag{7.16}$$

subject to

$$\sum_{k \in K} \sum_{i \in V^S} x_{i,j}^k = 1 \qquad \forall j \in V^N \tag{7.17}$$

$$\sum_{i \in V^S} x_{i,j}^k \leq 1 \qquad \forall j \in V^N, \forall k \in K \tag{7.18}$$

$$x_{i,j}^k \quad \textit{binary} \qquad \forall i, j \in V, \forall k \in K \tag{7.19}$$

We can see that both the generalised assignment problem and resource scheduling problem do not have conflict assignment constraints because they assume visits can be made anytime. Therefore, we decided to add the conflict assignment constraint to the generalised assignment problem because the number of constraints of the generalised assignment problem is less than that of the resource scheduling problem.

From the four components in Section 7.2.1, the compact MIP model is implemented by focusing particularly to reduce the number of constraints and the number of variables. We reformulate the model using only three constraint sets: assignment constraints, working hour limitation constraints, and conflict avoidance constraints. Comparing to the full model, constraints, such as route continuity, start-end location, minimum skill requirements, time windows, working region and time availability constraints, are applied during data pre-processing so that these constraints are considered when producing the conflict matrix, compatibility matrix and cost matrix.

$$\text{Minimise} \sum_{k \in K} \sum_{j \in T} c_j^k x_j^k + \sum_{j \in T} \lambda_4 y_j \tag{7.20}$$

subject to

$$\sum_{k \in K} b_j^k x_j^k + y_j = r_j \qquad \forall j \in T \tag{7.21}$$

$$\sum_{j \in T} \delta_j x_j^k \leq h^k \qquad \forall k \in K \tag{7.22}$$

$$(x_i^k + x_j^k) q_{i,j} \leq 1 \qquad \forall i, j \in \hat{T}(k), \forall k \in K \tag{7.23}$$

$$x_j^k \quad \textit{binary} \qquad \forall j \in T, \forall k \in K \tag{7.24}$$

$$y_j \quad \textit{integer} \qquad \forall j \in T, \forall k \in K \tag{7.25}$$

Note that

$$\hat{T}(k) = \{t | b_j^k = 1, \forall j \in T, b_j^k \in B\} \qquad , \forall k \in K \tag{7.26}$$

The members of the set $\hat{T}(k)$ are visits that the worker k can take where $b_j^k = 1$. This special set is used only in the conflict avoidance constraint (7.23) to reduce the number of generated constraints.

- **Visit Assignment Constraint**

  An assignment constraint is implemented in the same way as the full MIP model. Therefore, for every visit, the number of compatible workers and unassigned visits must equal the number of workforce required by a visit. This constraint is presented in (7.21).

- **Working Hour Limit Constraint**

  Working hour limitation is also implemented in the same way with the full MIP model to prevent assigning a worker more than the allowed working hours. Every worker has their own working hour limitation based on their working contract. The constraint is presented in (7.22)

- **Conflict Avoidance Constraint**

  Conflict avoidance constraint is tailored to the HHC scenarios based on the conflicting matrix. If two visits are conflicted time-wise, $q_{i,j} = 1$, at most one of the two visits can be assigned to the worker k as shown in (7.23). To reduce constraint redundancy, the conflict avoidance constraint is generated only if the worker k is compatible with the two visits i, j. We can see that the constraint is not required when the worker k is compatible with only one of the two visits, e.g. suppose $b_i^k = 1$ and $b_j^k = 0$, we can see that only $x_i^k = 1$ can fill workforce requirement in visit i while $b_j^k = 0$ in constraint (7.21), $x_j^k = 1$ does not fill the requirement of visit j. Thus, by minimising the objective function, the optimal solution to the compact model must have $x_j^k = 0$ which is then redundant to the conflict avoidance constraint.

The objective function (7.20) of the compact model has been simplified to only minimising assignment costs and unassigned visit costs. The assignment cost is provided by cost matrix C. The weight of the unassigned visits is set to λ_4 where the value is the same as the weight of unassigned visits in the full model.

Table 7.3 compares the implementation of constraints between the full model and the compact model. Both models are implemented as minimisation problems but with different types of variables: the full model as a network flow based model and the compact model as an assignment based model.

The compact model applies constraints in different ways compared with the full model. Constraints that have been applied when generating the conflict matrix are travel time feasibility and time window constraints. Conflict avoidance constraints (7.23) use the conflict matrix to guarantee that assignments made to the workforce are feasible, which include travel time feasibility and arriving time feasibility. The compatibility matrix is a result of merging con-

Table 7.3: Constraints implementation in the full model and the compact model.

	Full model	Compact model
Model type	Network flow based	Assignment based
Problem type	Minimisation	Minimisation
Constraints		
Visit assignment	Constraint (2.1), directed edges (from *i* to *j*)	Constraint (7.21), Assign workers with compatibility
Unassigned visit	Objective penalty tier 4	Objective penalty tier 4
Route continuity	Flow constraint (2.2)	Not in use
Start and end location	Constraint (2.3)- (2.6), Flexible start-end	Not in use
Skills and Qualifications	Constraint (2.9)	Merge with assignment using compatibility matrix and cost matrix tier 5
Contract	Constraint (2.9), As minimum skill constraint	Merge with assignment using compatibility matrix and cost matrix tier 5
Additional skills	Objective penalty tier 2	Cost matrix tier 2
Travel time feasibility	Constraint (2.7), direct interpretation	Constraint (7.23), conflicting matrix
Time window	Constraint (2.8)	Not in use, fixed time assignment
Working hour limit	Constraint (2.10)	Constraint (7.22)
Region availability	Constraint (2.13), Objective penalty tier 3	Cost matrix tier 3
Workforce time availability	Constraint (2.11)-(2.12), Objective penalty tier 3	Cost matrix tier 3
Travelling distance	Objective penalty tier 1	Est. cost matrix tier 1
Monetary cost	Objective penalty tier 1	Cost matrix tier 1
Workforce-visit preference	Objective penalty tier 2	Cost matrix tier 2
Region preference	Objective penalty tier 2	Cost matrix tier 2

tracts and skills and qualifications constraints. The cost matrix is generated by measuring a summation of constraint costs, the sum includes skills and qualifications penalties (tier 5), contract penalties (tier 5), region availability penalties (tier 3), workforce time availability penalties (tier 3), additional skill penalties (tier 2), workforce-visit preference penalties (tier 2), region preference penalties (tier 2), monetary costs (tier 1); and travelling distance estimation (tier 1). The related constraints in the full model are considered through the cost generation. The matrix is used in the objective function (7.20) where assignments that violate skills and qualifications and/or contract constraints are not made because leaving those visits unassigned is cheaper. Also, the backbone of the assignment problem is the visit assignment constraint (7.21) where it is formulated in a similar formulation to the full model, apart from the reduction of variable dimension. The working hour limit constraint (7.22) in the compact model is kept in the same format as the constraint in the full model. There are two constraints, which are route continuity; and start and end location, which are not required in the compact model because of differences between the two model structures. Both constraints generate the order of visits for every worker. However, the order of visits can be determined by the fixed assignment time.

A solution to the compact problem is in assignment format which is a list of assignments of workers to make visits. Generally, the solution of the assignment problem alone does not explicitly express a sequence of visits. However, this problem has assignments in fixed times so we can identify the visit sequences easily by sorting the assigned visits by arrival times. The solution with arrival times and sequences is then converted to the network flow based solution in order to calculate the real objective value where we use this value to compare results. The solution conversion is presented in the next section.

7.3 Solution Conversion

This part explains how a solution to the compact model can be mapped to a solution for the full model. We convert an assignment solution into a network flow solution. An assignment solution for the compact model can be simply defined by

$$\Phi_C = \{(k,j)|k \in K, j \in T \text{ and } x_j^k = 1\} \cup \{(0,j)|y_j = 1 \text{ and } j \in T\}.$$

An ordered pair (k,j) is an assignment of worker k to make visit j and assignment $(0,j)$ is an unassigned visit j.

However, a network flow solution for the full model has different structure defined by

$$\Phi = \{(k, i, j, a_j) | x^k_{i,j} = 1, a_j \geq 0, k \in K, i, j \in V\} \cup \{(0, j) | y_j = 1, j \in T\}.$$

An assignment is made by allocating a worker k to an edge linked between node i and node j and they must arrive at the visiting location at time a_j. Again, an unassigned visit j is denoted by $(0, j)$.

The conversion process starts from:

1. Generating a sequence of assignments $\Phi^k_C = \{(k, j)\}$ for a worker k to make visits $j = 1, \ldots, n$ such that $a(j-1) \leq a(j)$ when $a(j-1)$ and $a(j)$ are fixed assignment times of visits $j-1$ and j respectively (Grouped by worker and ordered by fixed assigned time);

2. Reading a sequence of assignments Φ^k_C and assigning edges by having $x^k_{j-1,j} = 1$ for each $(k, j-1), (k, j) \in \Phi^k_C$ from $j = 2, \ldots, n$;

3. Including the start and end nodes by $x^k_{d,j} = 1$ and $x^k_{n,d'} = 1$ when d is a starting location for worker k, j is the first location in the assignment sequence, n is the last assignment in the sequence and d' is the ending location for worker k;

4. Applying the visiting arrival time $a^k_j = a(j)$ if $x^k_j = 1$ where $a(j)$ is a fixed arrival time of visit j.

5. Applying the ending arrival time $a^k_{d'} = a(n) + \delta_n + t_{n,d'}$ where $a(n)$ is a fixed arrival time of last visit n, δ_n is a working duration at visit n, $t_{n,d'}$ is a travelling time between the last visit n and the ending location d';

6. Adding unassigned visit $y_j = 1$ to the solution.

The solution conversion generates paths for all workers in the full model solution format. The objective value is then evaluated to find the actual cost of the converted solution. The converted solution satisfies all full model constraints. Next, this chapter explains experiment and results comparing the compact model solution to the other solution methods.

7.4 Experiment and Results

7.4.1 Reformulation Performance Comparison with the Decomposition Approaches

Our experiment evaluates solutions obtained for the compact MIP model by comparing their objective values and computational times to the other solution methods. The data instances used in this experiment are the 42 HHC instances first presented in Chapter 2 since the compact model is tailor-made for the cases. The other solution methods are the geographical decomposition with conflict repair (GDCR), the repeated decomposition and conflict repair method (RDCR) and the heuristic algorithm (see in Chapter 5).

First, this experiment presents a number of constraints in the compact MIP model compared to the full model. Figure 7.1 presents the estimated number of constraints generated by both model implementations for the 42 HHC instances. The figure contains six sub-figures which represent six HHC scenarios and each scenario has seven instances. The graph shows that the full model generates a very high number of constraints, i.e. up to 25.4 billion constraints (2.7 billion constraints on average). As expected, the compact model produces fewer constraints in all instances. The number of constraints from the compact model ranges from 82 to 141 million constraints (17 million constraints on average). Thus, the compact model generates less than 4.23% of the number of constraints expected from the full model.

Reducing the number of constraints leads to a major reduction in memory

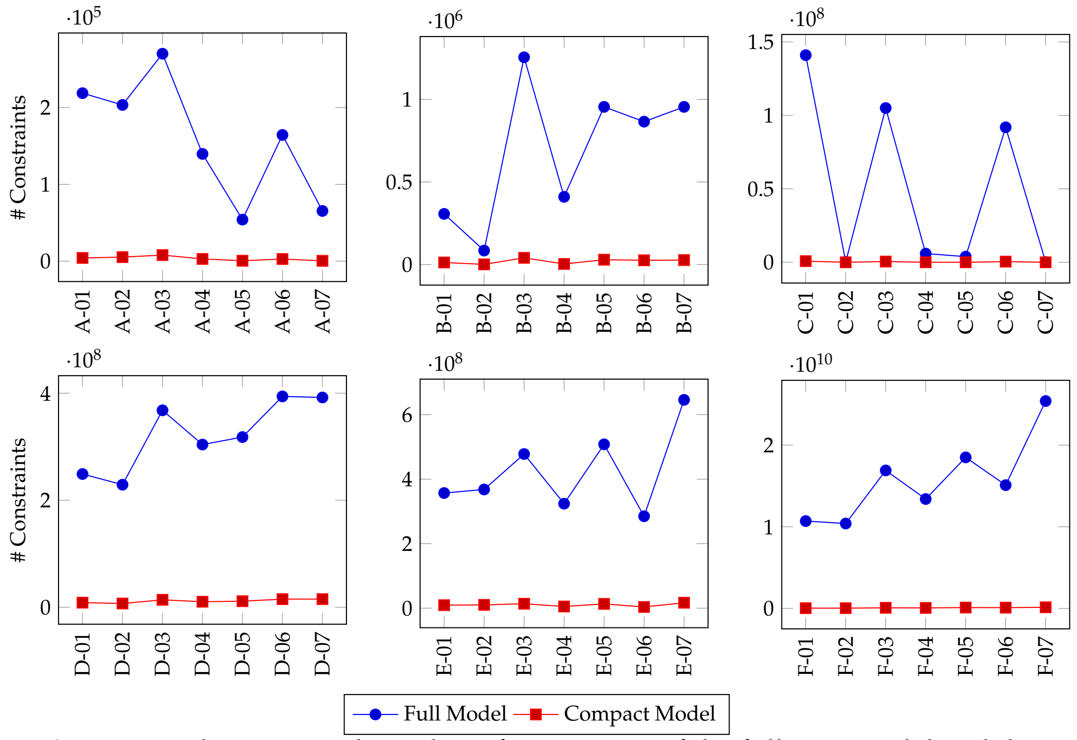


Figure 7.1: The estimated number of constraints of the full MIP model and the compact MIP model. The figure contains six sub-figures each for a different scenario dataset.

required by CPLEX to solve such a problem instance. The memory requirement is estimated from the number of the constraints. From CPLEX manuals, 1 MB is required for every 1,000 constraints [2]. For example, the largest instance (F-07), the compact model has an estimated memory requirement of 135 GB while the full model has an estimated memory up to 24,839 GB. From the memory estimation, only instance set F has memory estimation of more than 16 GB which exceeds the amount of RAM of the standard personal computer used in our experiments. Therefore, we solved the compact problem using two different environments: a desktop PC and a high performance computing machine (HPC). The HPC is a cluster of high specification computer servers provided by the University of Nottingham High Performance Computing facility. The computing resources we use in this experiment are enhanced computing nodes with 16 computing core with 128GB of RAM.

The solution is then observed mainly on solution objective function. Table

Table 7.4: Objective value of solutions provided by five solution methods.

Instance	Optimal	GDCA	GDCR	RDCR	Heuristic	Compact
A-01	**3.49**	5.65	4.48	4.00	5.46	3.49*
A-02	**2.49**	4.53	3.36	2.93	4.42	**2.49**
A-03	**3.00**	10.6	4.93	6.86	7.41	**3.00**
A-04	**1.42**	3.09	2.49	1.95	2.36	1.42*
A-05	**2.42**	3.54	3.12	2.42	3.15	**2.42**
A-06	**3.55**	3.74	3.62	3.56	5.67	3.55*
A-07	**3.71**	4.81	4.07	3.71	7.10	**3.71**
B-01	**1.70**	1.79	1.89	1.76	2.87	**1.70**
B-02	**1.75**	1.89	1.75	1.80	2.83	**1.75**
B-03	**1.72**	2.06	1.89	1.85	2.97	**1.72**
B-04	**2.07**	2.21	2.13	2.12	3.01	**2.07**
B-05	**1.82**	4.74	2.54	2.98	2.88	**1.82**
B-06	**1.62**	2.52	1.75	1.75	2.91	1.62*
B-07	**1.79**	4.06	2.94	2.03	3.44	1.79*
C-01	N/K	905	133	132	185	**115**
C-02	**3.15**	3.61	3.15	3.15	4.86	**3.15**
C-03	N/K	1,186	196	159	170	**105**
C-04	**11.15**	81.3	23.07	13.08	16.51	**11.15**
C-05	**12.34**	68.9	22.39	15.25	17.59	**12.34**
C-06	N/K	3,102	198	197	251	**179**
C-07	**4.30**	5.29	**4.30**	**4.30**	5.82	**4.30**
D-01	N/K	496	210	205	236	**170**
D-02	N/K	373	206	199	244	**162**
D-03	N/K	3,213	229	208	221	**178**
D-04	N/K	419	219	212	223	**166**
D-05	N/K	244	202	184	189	**162**
D-06	N/K	1,411	223	199	199	**177**
D-07	N/K	753	218	203	197	**179**
E-01	N/K	33.0	3.69	5.19	6.27	**1.16**
E-02	N/K	26.0	2.21	3.22	4.83	**1.17**
E-03	N/K	29.0	1.23	4.23	8.27	**1.19**
E-04	N/K	28.5	1.79	1.79	4.30	**1.27**
E-05	N/K	270	3.76	7.26	8.25	**2.22**
E-06	N/K	24.6	2.30	2.30	5.43	**1.29**
E-07	N/K	428	4.72	7.71	5.68	**1.70**
F-01	N/K	64,305	2,740	2,150	2,810	**2,010**
F-02	N/K	73,291	2,482	2,505	3,235	**2,069**
F-03	N/K	115,235	707	704	1,619	**597**
F-04	N/K	102,994	1,453	1,448	1,958	**1,321**
F-05	N/K	101,438	297	315	1,752	**178**
F-06	N/K	76,007	747	742	862	**616**
F-07	N/K	176,541	3,610	3,604	4,239	**3,484**

Bold text refers to the best solution.
N/K is for solution currently not known.
* solution is marginally different to the optimal solution.

7.4 presents the objective values for six solution methods: the full model (Optimal), GDCA, GDCR, RDCR, Heuristic assignment and the compact model (Compact). Note that the lower objective value is the better solution.

From table 7.4, the best solution value is presented in **bold** text. We note that an optimal solution is available only when the instance is small enough to be solved. An optimal solution is found only for 18 instances (Sets A, B and four instances of C) from solving the full model. Among the 18 instances for which the optimal solution can be found, the reformulation approach finds a solution of matching quality for 13 of those instances. The objective value of the other 5 solutions is marginally different to that of the corresponding optimal solution ($<$1% relative gap). For the other solutions where the optimum is not known, the compact model solutions are the best known solutions so far. These results show that the compact model representation is very effective.

In addition, we apply related-samples Friedman's two-way analysis of variance to test differences between the objective values provided by the four solution methods. The result of this analysis show that distributions between the methods are significantly different with test statistic at 138.91 and p. value less than 0.01. The mean rank of compact MIP model solution is 1.08. Pairwise comparisons show that solutions found with the compact MIP model are significantly different to the other three approaches.

Table 7.5 presents computational times used by the six solution approaches. The computational times were recorded in seconds where the **bold** presents the lowest computational time for that instance. The quickest approach is clearly the heuristic assignment algorithm where it is the quickest algorithm on every instance. The second fastest algorithm is labelled by an "*" (asterisk) mark. The results in Table 7.5 reveal that the compact model was the third fastest algorithm where the RDCR was the second fastest approach. We can see that the RDCR had the second lowest computational time on 26 instances while the

Table 7.5: Computational time (seconds) for solving a solution using six solution methods.

Instance	Optimal	GDCA	GDCR	RDCR	Heuristic	Compact
A-01	7.00	3.71	3.76	2.53	**<0.01**	2.11*
A-02	8.00	3.58	3.35	2.39	**<0.01**	0.15*
A-03	14.00	3.70	4.69	5.17	**<0.01**	0.16*
A-04	5.00	2.88	2.29	1.87	**<0.01**	0.12*
A-05	1.00	1.77	1.28	0.76	**<0.01**	0.14*
A-06	5.00	2.42	2.80	1.77	**<0.01**	0.13*
A-07	1.00	1.64	1.55	0.70	**<0.01**	0.10*
B-01	21.00	8.07	6.96	4.67	**<0.01**	0.16*
B-02	2.00	4.29	3.36	0.79	**<0.01**	0.11*
B-03	6,003	32.9	38	11	**<0.01**	0.29*
B-04	25	15.2	12	2.63	**<0.01**	0.13*
B-05	585	25.3	23	8.31	**<0.01**	0.25*
B-06	184	24.1	22	8.78	**<0.01**	0.22*
B-07	300	23.6	24	8.14	**<0.01**	0.24*
C-01	N/K	212	224	26*	**0.34**	51
C-02	6.00	0.57	0.63	0.12*	**<0.01**	0.21
C-03	N/K	26.33	28	18*	**0.26**	38
C-04	90	3.09	3.84	1.07*	**0.11**	1.74
C-05	55	1.05	1.91	0.71*	**<0.01**	1.09
C-06	N/K	47.0	50	25*	**0.19**	33
C-07	1.00	0.24	0.23	0.11*	**<0.01**	0.13
D-01	N/K	1,060	579	109*	**0.18**	253
D-02	N/K	1,192	706	109*	**0.14**	262
D-03	N/K	1,209	1,024	127*	**0.18**	544
D-04	N/K	3,005	785	127*	**0.17**	419
D-05	N/K	1,307	907	118*	**0.18**	206
D-06	N/K	1,222	1,064	130*	**0.20**	304
D-07	N/K	1,362	1,133	142*	**0.23**	325
E-01	N/K	8,408	7,676	94*	**0.19**	112
E-02	N/K	12,448	9,806	87*	**0.18**	129
E-03	N/K	20,747	11,872	101*	**0.22**	196
E-04	N/K	15,190	8,758	72	**0.18**	53*
E-05	N/K	32,619	9,510	99*	**0.25**	181
E-06	N/K	24,212	9,121	66	**0.15**	33*
E-07	N/K	51,057	13,884	107*	**0.27**	271
F-01	N/K	3,446	1,788	250*	**1.00**	730
F-02	N/K	1,111	1,730	251*	**1.20**	767
F-03	N/K	4,555	1,908	342*	**1.61**	2,119
F-04	N/K	4,219	7,060	360*	**1.67**	2,040
F-05	N/K	6,157	3,437	390*	**1.91**	3,428
F-06	N/K	9,696	7,204	442*	**1.91**	2,628
F-07	N/K	3,833	1,847	422*	**2.46**	3,570

Bold text refers to the fastest computational time.
N/K is for solution currently not known.
* the second fastest computational time.

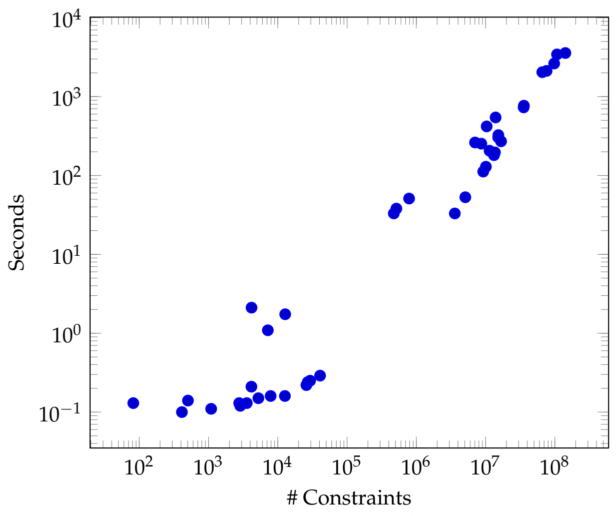


Figure 7.2: Scatter Plot between the number of constraints and the computational times to solve the compact model. Both axes are in logarithmic scales.

compact model had the second lowest computational times on 16 instances. The instances that the compact model used less computational time than the RDCR are sets A, B, and instances E-04 and E-06. In short, the RDCR is the faster approach comparing to the compact model.

For a closer look, the compact model is the second quickest on instance A and B where the method use less than a second to solve 13 instances, only A-01 required 2.11 seconds to find the solution. The RDCR computational times were ranged from 0.76 seconds to 8.78 seconds on the same sets. For the larger instance sets D, E, and F, the compact model used double the RDCR computational times on instance set D and at least triple the RDCR times on instance set F but both methods showed comparable times on set E.

Figure 7.2 presents a scatter plot between the number of constraints and the computational times (in seconds) of the compact model in logarithmic scales. The graph shows a linear relation between the number of constraints and computational times. However, the number of constraints is growing exponentially when compared to the problem size. Again, this graph shows that the lower the number of constraints the less computational time required to solve the problem.

We apply statistical test to validate our results on both solution quality and computational time. Table 7.6 presents statistical results from related-samples Friedman's ANOVA to find differences on objective value and computational time between five solution approaches. The statistic did not take the full model approach into account because there were solutions that have not yet known. Each test reports the calculated statistics and mean ranks for each of five methods.

For the test on objective value, the statistical test shows the calculated statistics $\chi^2 = 138.91$ which give p-value$< .01$. This means the objective values between five methods are different at significant level $\alpha = .05$. The mean ranks show that the compact model provides the best method to find the best result with mean rank at 1.08. The second best approach is the RDCR where the mean rank is 2.33, followed by the GDCR with 2.88 mean rank, heuristic assignment algorithm with 3.94 mean rank, and the GDCA at 4.76 mean rank.

For the test on computational time, the statistical test shows the calculated statistics $\chi^2 = 142.64$ which give p-value$< .01$. Therefore, the computational times between five methods are different at significant level $\alpha = .05$. The mean ranks show that the heuristic assignment algorithm is the fastest method where the mean rank is 1.00. The second fastest approach is RDCR with 2.43 mean rank, followed by the compact model at 2.74 mean rank, the GDCR at 4.21 mean rank, and the GDCA at 4.62 mean rank.

The statistical tests conclude that the approach to find the best solution is the compact model and the fastest method is the heuristic assignment algorithm. Overall, the compact model and the RDCR are approaches for both good solution quality and less computational time.

Table 7.6: Friedman statistical test on solution quality and computational time on five solution methods.

Objective value				Computational time			
Friedman Test		Mean Ranks		Friedman Test		Mean Ranks	
N	42	GDCA	4.76	N	42	GDCA	4.62
χ^2	138.91	GDCR	2.88	χ^2	142.64	GDCR	4.21
df	4	RDCR	2.33	df	4	RDCR	2.43
p	<.01	Heuristic	3.94	p	<.01	Heuristic	1.00
		Compact	1.08			Compact	2.74

7.4.2 Reformulation Performance Comparison with Other Heuristic Algorithms

The other heuristic approaches which have been used to solve the HHC instances are a variable neighbourhood search algorithm (VNS) [103] and a genetic algorithm (GA) [6]. This section summarises the two algorithms and then compares the compact model solution to the solutions of the two heuristic approaches. These results have been kindly provided by the paper authors and were generated using a personal computer with the same specification as our computing machines. We note that there are different computer specs to solve instance set F, i.e. the compact model was solved by HPC but other algorithms were run on the PCs.

Variable Neighbourhood Search to Solve the HHC

A variable neighbourhood search (VNS) to solve the home healthcare planning was implemented by Pinheiro et al. [103]. The VNS has two iterative stages: a shaking phase and a local search phase. The shaking phase randomly selects one of seven shaking neighbourhoods. If changes cannot be made to the solution, another shaking neighbourhood is selected. The process is repeated until a change is made to the solution. Then, the local search phase takes the changed solution to generate neighbouring solutions by using two neighbourhood search operators. The two operators should deliver improved neighbouring solutions. The local search is applied until no further improvements can be made. At this point, the algorithm goes to the shaking phase. The two phases are executed in an iterative manner until the algorithm reaches the time limit, e.g. one hour. For more detail, see [103].

Genetic Algorithm to Solve the Home Healthcare Problem

A genetic algorithm (GA) to solve the home healthcare planning was proposed by Algethami and Landa-Silva [6]. The GA is implemented with a simple direct representation scheme and an uniform mutation crossover. The algorithm sets a mutation rate at $1/|T|$ [68]. A 100-individual population is selected by tournament selection where 10% of the best individuals are always kept on the offspring population. The algorithm avoids getting stuck in local optima and early convergence condition by using a reset mechanism. After the GA cannot find improvement after 10 generations, the reset mechanism generates the bottom half of the population randomly to increase the diversity of the population. For more detail, see [6].

7.4.3 Results and Discussions

This section compares objective value of solutions produced with the proposed problem reformulation approach to the solutions produced with the two selected heuristic algorithms: VNS and GA. The optimal solutions (when known) are also shown. Table 7.7 presents the objective value of the solutions obtained by the four solution methods. The best known solution is highlighted in **Bold**. In addition, solutions which have objective value within 1% relative gap to the best known solution are highlighted with an “*" (asterisk) mark.

The results show that the solutions of the compact model are best known solutions for 29 instances. This is followed by the VNS with best known solutions for 23 instances and then the GA which finds 8 best known solutions.

Table 7.7: Comparison of objective values between optimal solution, variable neighbourhood search (VNS), genetic algorithm (GA) and compact model solution (Compact).

Set	Optimal	VNS	GA	Compact
A-01	**3.49***	**3.49***	**3.49***	3.49*
A-02	**2.49***	**2.49***	**2.49***	**2.49***
A-03	**3.00***	3.00*	3.02	**3.00***
A-04	**1.42***	**1.42***	1.42*	1.42*
A-05	**2.42***	**2.42***	**2.42***	**2.42***
A-06	**3.55***	**3.55***	3.55*	3.55*
A-07	**3.71***	**3.71***	**3.71***	**3.71***
B-01	**1.70***	**1.70***	1.74	1.70*
B-02	**1.75***	**1.75***	**1.75***	**1.75***
B-03	**1.72***	1.72*	1.78	1.72*
B-04	**2.07***	**2.07***	2.08	**2.07***
B-05	**1.82***	1.84	1.92	**1.82***
B-06	**1.62***	1.62*	1.64	**1.62***
B-07	**1.79***	1.79*	1.80*	1.79*
C-01	N/K	**114.21***	114.24*	**114.21***
C-02	**3.15***	**3.15***	**3.15***	**3.15***
C-03	N/K	103.52*	104.00*	**103.52***
C-04	**11.15***	**11.15***	**11.15***	**11.15***
C-05	**12.34***	**12.34***	12.60	**12.34***
C-06	N/K	**140.44***	141.37*	**140.44***
C-07	**4.30***	**4.30***	**4.30***	**4.30***
D-01	N/K	**168.20***	171.71	170.37
D-02	N/K	**161.23***	167.49	161.89*
D-03	N/K	**177.83***	180.80	178.47*
D-04	N/K	**165.45***	169.80	166.38*
D-05	N/K	**161.08***	161.86*	162.20*
D-06	N/K	177.38*	178.16*	**177.38***
D-07	N/K	**177.76***	178.71*	178.88*
E-01	N/K	1.17*	1.18	**1.16***
E-02	N/K	1.17*	1.21	**1.17***
E-03	N/K	1.20*	1.22	**1.19***
E-04	N/K	1.27*	1.30	**1.27***
E-05	N/K	2.23*	2.24	**2.22***
E-06	N/K	1.29*	1.30	**1.29***
E-07	N/K	1.70*	1.73	**1.70***
F-01	N/K	2,011.21*	2,592.56	**2,009.83***
F-02	N/K	2,071.80*	2,652.67	**2,068.67***
F-03	N/K	599.48*	1,372.24	**596.78***
F-04	N/K	**1,319.86***	2,064.93	1,320.86*
F-05	N/K	182.46	1,092.35	**177.83***
F-06	N/K	618.23*	1,439.86	**615.85***
F-07	N/K	3,485.19*	4,419.73	**3,483.93***

Bold text presents the best known solution.
N/K is for solution currently not known.
* Objective value is less than 1% gap to the best known.

In addition, the number of solutions where the objective value is within 1% relative gap of the compact model, the VNS and the GA are 41 solutions, 40 solutions and 19 solutions, respectively. This is an indication that the VNS and the compact model deliver solutions with the same quality.

For instance set C, the solution of the reformulation approach can be acknowledged as an optimal solution because the problem instances do not have distances to travel between locations which results in the exact cost matrix. Therefore, the optimal solution of the reformulated model is the optimal solution of the full problem.

The other instances which require the traveling distance estimation, the reformulation approach finds the best known solutions in most cases except instances in set D. The instance set D demonstrates the deviation of the objective direction when applying the estimated travelling distances. This results in a decrese in the solution quality as shown in instance set D. However, the objective direction might not have strong effect in instance set E and F as the result shows the compact model retains the best known solutions.

7.5 Summary

This chapter presents a reformulation technique to solve the HHC problem. The reformulation is made based on requirements of the HHC scenarios. Thus, the HHC problem, which originally is formulated as a flow based model, is reformulated into a compact model which is defined as an assignment problem with restriction on conflicting visits i.e. assignment of a worker to make no more than one conflicting visit. Note that conflicting visits are visits that are time-wise overlapped, where a worker can make at most one visit amongst the conflicting visits. The compact model reduces over 95% of the total number of constraints generated by the full model, which results in the reduction of solver computational memory requirements.

The compact model can be solved by the mathematical solver without applying decomposition techniques. Most instances were solved by a standard PC, except instances in set F which requires high performance computing machines. The solution of the compact model has the best objective value amongst all the decomposition/reformulation approaches. The statistic also confirmed that using the compact model is the best solution approach with significant differences.

The computational time to solve an instance implemented in the compact model is less than 1 hour (about 7 minutes on average). Although, the average computational time ranks third out of five solution approaches, we consider the solving time to be acceptable (less than 8 hours).

The experiment also compares the reformulation approach to a variable neighbourhood search and a genetic algorithm also developed specifically to tackle these HHC instances. There are 30 solutions from the reformulation approach are the best known solutions to date. If the solution is not the best known, the objective value is only within 1% relative gap from the best known solution.

From our study, the compact model still has some limitations in solving the instance set F. This is because the compact model memory requirements can still be very large for using standard PCs. We may conclude that the instance set F could be the largest solvable instances by the compact model. Thus, instances larger than F-07 should be tackled by other methods such as decomposition approaches or heuristic methods.

In addition, the compact model is defined specifically to the HHC problem. Research would be required to determine how to extend the approach to the generalised WSRP. Technically, the compact model works because of the fixed visiting time which then leads to the design of the conflicting matrix. However, the assigned time on other WSRP instances might be flexible, e.g. given by time windows. Therefore, implementing a conflicting matrix for the time window problem should be investigated in future work.

Chapter 8

Conclusions and Future Work

8.1 Summary of Work

This thesis studies approaches to harness the use of an MIP solver in order to tackle real-world HHC instances and some benchmark WSRP instances. The main set of instances used in this research is a set of real-world HHC instances that have specific features related to soft conditions on worker time availabilities and worker region availabilities. The instances are grouped in six scenarios, labelled as A, B, C, D, E, and F, each of which has 7 instances. The HHC instances have a number of visits in the range from 6 to 1,726 visits and the number of workers in the range from 19 to 1,077 workers. The second set of the instances used in this thesis is a set of WSRPs with time-dependent activities constraints. The benchmark WSRP instances are presented in four scenarios, labelled as Sec, Sol, HHC, and Mov. The first scenario, Sec, is the case of security guard patrols which has 180 instances. The second scenario, Sol, is a group of 168 adapted Solomon instances modified by adding time-dependent activities, in the work by Castillo-Salazar et al. [35]. The third scenario, HHC, is a group of 11 home healthcare instances provided by Rasmussen et al. [107]. The last group, Mov, is a group of 15 instances of the vehicle routing problem with time windows described in Castro-Gutierrez et al. [37]. The total number of the benchmark WSRP instances is 374. These benchmark instances have been used in order to investigate the performance of the proposed decomposition approach when tackles problems with time windows and time-dependent activities.

A MIP model was proposed in Chapter 2 to formulate the problem with 42 HHC instances provided by our industrial partner used for computational testing. Finding the optimal solution for each of the 42 HHC instances is very challenging, particularly the larger instances in sets D, E, and F. The MIP solver can find the optimal solution for 18 smaller instances, 14 from sets A, B and four instances from set C. The solver ran out of memory when tackling the other 24 instances. Each of the 18 instances solved to optimality had estimated memory requirement less than 16 GB, which is the memory size of the PC used in this research. On the other hand, the 24 larger instances had estimated memory requirement to be much higher than 16 GB and as much as 24.8 TB. The results and the estimated computing resources indicate that using the MIP solver to tackle large real-world instances is not practical.

The first part of Chapter 3 presents a traditional decomposition approach to avoid the MIP solver using the full amount of required memory. The column generation algorithm is a traditional decomposition approach where smaller parts of the problem are provided to the solver at the beginning of the process, which is the key to reduce memory requirements. Later on, additional parts of the problem are added systematically until the algorithm reaches an optimal condition or there are no columns that could improve the current solution. Dantzig-Wolfe decomposition is the main concept behind the column generation algorithm allowing a problem to be decomposed into a master problem and multiple sub-problems. The experiment which tackled the HHC instances with column generation algorithm is described in Chapter 3. The experiment shows that the column generation algorithm take very long computational time. The experiment also shows that an iteration to process the largest instance, C-07, cannot be completed within 2 hours. The column generation algorithm can find the optimal solution for each of the 18 smaller instances, but computational times are significantly higher than when solving the instance as a whole. Besides, solutions for the 24 larger instances have not yet been found by this algorithm. However, the study revealed that using the MIP solver to solve sub-problems is possible. The weakness of this approach is the computation time, particularly when solving sub-problems in every iteration.

From the problem difficulty we learnt from Chapter 2 and the first part of Chapter 3, our research changed focus to heuristic decomposition approaches where the aim is to find a good feasible solution. Thus, the second part of Chapter 3 reviews heuristic decomposition approaches implemented in the literature. There are several approaches in the literature such as algorithms inspired by the traditional decomposition, methods to partition problem and solve sub-problems as independent problem, and methods to group problem elements. An approach chosen partitions the problem and then solves sub-problems independently was an implementation. The strong points are that all decisions can be made by the solver and that the sub-problem size is adjustable which give an ability to control solver memory requirements. However, the solution quality of this approach is dependent on the partitioning method used. Thus, this thesis proposes two main decomposition approaches: a conflict avoidance approach and a conflict repair approach.

The conflict avoidance approach prioritises sub-problems to access to their resources, in this case is workforce. This thesis proposes a geographical decomposition with conflict avoidance (GDCA) method in Chapter 4. The algorithm splits a problem using geographical data and prioritise sub-problems as the conflict avoidance approach. The result shows that GDCA can find a feasible solution for every HHC instance using a standard PC. This algorithm has delivered the first set of feasible solutions for the 24 larger instances.

The solutions produced by the GDCA can be improved by reducing unassigned visits. The improved version of the GDCAis proposed, where sub-problems may use neighbour workforce from its nearby geographical regions. The solution improved by algorithm should have more assignments as neighbour workforce is utilised. As a result, the unassigned visits are reduced and the objective function improves by 40% of relative gap on average. However, the solution quality is very dependent on sub-problem solving sequences, but finding the best sequence is not practical.

Chapter 5 proposes algorithms where solving sequences are no longer needed. During the decomposition stage, this approach allows every sub-problem to have access to all available workers. After solving the sub-problems, the solutions have conflicting assignments, e.g. two assignments that overlap in time and are assigned to the same worker. *Conflicting assignments repair* is a process to deal with these conflicting assignments with solutions where the process may either rearrange assignments or remove some assignments to solve the conflicts. The remaining unassigned visits are then tackled with either a heuristic assignment or an iteratively decomposition and conflicting assignments repair process.

The first variant of the decomposition and repair method, a geographical decomposition with conflict repair (GDCR), is an algorithm to decompose a problem by geographical regions, then solve each sub-problem with full workforce availability and then tackle conflicting assignments by using a conflicting assignments repair process. The implementation of this approach solves the 42 HHC instances, where visits do not have time windows. Therefore, the decision to be made in the conflicting assignment repair process is to decide assignments to be removed and become unassigned visits. These unassigned visits are then tackled by a heuristic assignment algorithm which assigns visits to the cheapest available worker. The experiment shows that GDCR solutions (24% gap on average) are significantly improved from GDCA solutions (66% gap on average), with 42% gap differences. The analysis shows that a solution part by made heuristic assignment has slightly lower solution quality than assignments made by the MIP solver in solving the decomposition sub-problems step and the conflicting assignments repair step. The result of this analysis suggests another variant of the algorithm with conflict repair.

The second variant, a repeated decomposition with conflict repair (RDCR), is an algorithm to decompose a problem and repair conflicting assignments repeatedly until there are no unassigned visits or the number of unassigned visits is not decreased. Sub-problems are solved without preventing conflicting assignments in the same way with GDCR and conflicting assignments repair tackles these conflicting assignments. To decrease the overall computational time, a limit on the size of sub-problem is set. We expected that the objective value of solution to the overall problem might increase. However, the iterative process reduced the objective value to the same level as with GDCR solution. In fact, the solutions provided by RDCR are slightly better than with GDCR where differences in the average gap to the best known is 1.6%. Moreover, RDCR is the decomposition approach that takes the least computational time. The average computational time to solve an instance is less than 2 minutes while the maximum computational time is less than 8 minutes.

The result of RDCR is shown to be the best so far of the heuristic decomposition methods based on the full model implementation. This research investigates further application of the RDCR to solve the WSRP with time-dependent activities constraints in Chapter 6. A modification is made to the RDCR so that the solution satisfy the time-dependent activities constraints. An experiment compares RDCR performance to a greedy heuristic method, implemented by

Castillo-Salazar et al. [36]. The result show that RDCR provides a higher number of solutions in which their quality is better than the greedy heuristic solution. However, the differences are not statistically significant.

The research is extended to study performances of a small mathematical formulation which is tailor-made for the 42 HHC instances, as presented in Chapter 7. The compact MIP model presented in Chapter 7 has a lot fewer computational resource requirements as the estimated physical memory is only 1% of the full model. Most of the compact model solutions are the best known solutions so far. This showes the benefit of using the MIP solver over the other heuristic methods. However, implementing a compact formulation is very problem-specific which means the implementer must have insight into the problem.

Table 8.1 summarises the relative gap of solutions to the best known solution of the four techniques proposed in this thesis. From the table, the number of the best known solutions found by the compact model, RDCR, GDCR, and GDCA are 32, 4, 3, and 0 solutions, respectively. The average solution relative gaps to the best known of the compact model, RDCR, GDCR, and GDCR are <0.01, 22.05, 23.72, and 65.94, respectively. The result shows that the compact model is clearly the best approach to providing the highest solution quality.

To summarise, this thesis presents two different ways to tackle real-world instances and benchmark problems. First, heuristic decomposition methods tackle a problem by splitting it into sub-problems and solving each sub-problem using the full model formulations. This approach is easily applied to any problem which is designed as a graph and balanced flow structures. If nodes should be related, like time-dependent activities constraints, this can be tackled by RDCR with modification. However, the heuristic decomposition methods do not support scenarios where all nodes must be visited. The second approach, problem reformulation, is a fast and efficient method for finding the best quality

Table 8.1: Solution relative gap to the best known solution of four decomposition/reformulation techniques.

Instance	GDCA	GDCR	RDCR	Compact
A-01	38.29	22.18	12.76	<0.01
A-02	45.04	25.90	14.94	0.00
A-03	71.87	39.30	56.36	0.00
A-04	54.04	43.01	27.07	0.04
A-05	31.60	22.44	0.00	0.00
A-06	5.19	1.90	0.29	<0.01
A-07	22.74	8.65	0.00	0.00
B-01	4.82	10.05	3.49	0.00
B-02	7.31	0.00	2.73	0.00
B-03	16.51	9.32	6.90	0.01
B-04	6.28	2.50	2.34	0.00
B-05	61.55	28.10	38.90	0.00
B-06	35.60	7.23	7.52	<0.01
B-07	55.87	39.12	11.80	<0.01
C-01	87.32	13.45	12.81	0.00
C-02	12.63	0.00	0.00	0.00
C-03	91.19	46.78	34.44	0.00
C-04	86.28	51.67	14.79	0.00
C-05	82.10	44.89	19.06	0.00
C-06	94.22	9.55	8.87	0.00
C-07	18.68	0.00	0.00	0.00
D-01	65.68	18.76	17.00	0.00
D-02	56.59	21.31	18.56	0.00
D-03	94.45	21.99	14.15	0.00
D-04	60.28	23.93	21.44	0.00
D-05	33.49	19.55	11.72	0.00
D-06	87.43	20.55	10.82	0.00
D-07	76.25	18.04	11.82	0.00
E-01	96.47	68.42	77.56	0.00
E-02	95.51	47.09	63.68	0.00
E-03	95.91	3.49	71.99	0.00
E-04	95.54	29.01	29.05	0.00
E-05	99.18	40.95	69.42	0.00
E-06	94.77	43.99	44.15	0.00
E-07	99.60	64.05	77.99	0.00
F-01	96.87	26.65	6.51	0.00
F-02	97.18	16.66	17.43	0.00
F-03	99.48	15.56	15.21	0.00
F-04	98.72	9.11	8.77	0.00
F-05	99.82	40.08	43.52	0.00
F-06	99.19	17.59	17.04	0.00
F-07	98.03	3.50	3.34	0.00
Average	65.94	23.72	22.05	<0.01

solution. However, implementing the model may become difficult depending on constraint types and problem structure. This approach reduces the solver memory requirement down to only 1% of the full model, but the memory still could be very high for a normal computer when tackling the very large real-world instances, e.g. set F.

8.2 Future Work

Based on the studies in this thesis, we suggest future work on mathematical models, decomposition approaches, reformulation approaches, and workforce scheduling and routing problem.

8.2.1 Future Work on Mathematical Models

An MIP model is proposed in Chapter 2. The model has nine constraint types to cover requirements from real-world HHC problems. However, other real-world scenarios could have other requirements for which formulations are needed. Examples of these constraints are balancing workload, breaks between shift, overtime duration, temporally teaming skills, multiple transportation modality, etc.

However, implementing every constraint in one model may not be possible because some requirements can conflict, e.g. temporally teaming skills requirement can be conflicting with the workforce skill requirement. The temporally team skills are skills of the whole team where a worker may fulfil a subset of team skills and the team can make a visit if the team skills exceed the visit skill requirements. On the other hand, the workforce skill requirement specifies that every worker who makes a visit should be qualified. For comparison, a team from the model with teaming skills requirements cannot make a visit in the problem with the workforce skill requirements. Therefore, the deployment of both constraints in a single model may result in constraint redundancy; in this case, workforce skills requirement is tighter than teaming skills requirement. Therefore, a study and analysis of constraint implementation can be interesting for future research in order to build a model to cover all possible HHC problem requirements.

8.2.2 Future Work on Decomposition Approaches

Strong points of decomposition approaches are that their memory requirements are controllable by adjusting sub-problem size and their computational times were low. We can see that the solution quality has been improved from GDCA, GDCR to RDCR. Future research should continue improving the RDCR because it is the fastest algorithm that also provides high quality solutions. One suggestion is to explore new decomposition rules such as applying capacitated clustering algorithms. However, the capacitated clustering problem is another hard problem, particularly when tackling large real-world instances. Therefore, we suggest using heuristic clustering algorithms instead of solving the clustering exactly because clustering is not the basis of the problem. Examples of heuristic algorithm which have been applied to solve capacitated clustering problem are tabu search and simulated annealing [100]. Alternatively, the capacitated clusters can be generated by a column generation approach [90].

This thesis designs an implementation of RDCR to solve WSRP with time-dependent activities constraints. Future work is to improve RDCR performance when tackling this problem with different methods to partition the set of visits or using different workforce selection rules. Another approach to improve RDCR to tackle time-dependent activities constraints is to modify the conflicting assignments repair. Originally, conflicting assignments repair builds sub-problems, which are defined based on a single worker to resolve these conflicting assignments. The suggested approach is to build a conflicting sub-problem from several workers who have time-dependent activities. This is to replace the fixed time modification of the conflicting assignments repair. Alternatively, future work may develop an approach which replaces a fixed time modification with a restricted time window where the restricted value is related to the assignment time, the visit time window, and the visit time-dependent activities. The aim of replacement is to allow the conflicting assignments repair to move assignments to find a better solution and still satisfy the time-dependent activities constraint.

8.2.3 Future Work on Reformulation Approaches

The model reformulation to solve HHC showed a successful approach to tackle the problem to achieve the best results. However, we make a few suggestions for improvement in future work. The first issue is the memory consumption of the compact model because the memory requirement to solve instance set F exceeds the memory in a standard PC (16 GB). Therefore, future research should focus on constraint reductions. The constraint reduction can be done by creating small clusters of visiting nodes and using a cluster as a representative visit. This approach has been done by Dondo and Cerdá [56].

In addition, future work should extend the reformulation technique to tackle the WSRP. The main challenge is to deal with non-static conflict matrix because the existence of the time-window, i.e. two visits have the possibility to conflict when the time-window of both visits overlaps. We note that this future work should have an impact on wider audiences because the solution method can be applied to the other related problems, such as the VRP.

8.2.4 Future Work on Workforce Scheduling and Routing Problem

This thesis investigated the daily WSRP. Another variant of the problem is to consider multiple working days, or a weekly plan. Research in this direction should consider rostering constraints such as a worker can work at most two consecutive shifts, at most two night shifts in a row can be made by one worker, there must be two consecutive days-off in a week, maximum working hours in a week, etc. These constraints are not the only complicated part, but also the model concept must support multiple trips feature. The possible approach is to add assignment decision variable dimension which results in a higher memory requirement for the solver to tackle such an instance. In fact, there are a few publications on the weekly planing horizon in the literature [33, 34].

Finally, other future research is to investigate the multi-objective nature of the WSRP. The current work, presented in this thesis, used a weighted-sum objective function which gives priority to the visit assignment over preferences or cost. However, other real-world applications might have different objective priority setup. Therefore, a multi-objective approach will give choice to the decision maker when tackling a problem with different objective priorities.

Bibliography

[1] IBM ILOG CPLEX optimization studio. `http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud`, 2013. Accessed: 2016-02-13.

[2] Guidelines for estimating CPLEX memory requirements based on problem size, 2016. URL `http://www-01.ibm.com/support/docview.wss?uid=swg21399933`.

[3] K. Aardal. Reformulation of capacitated facility location problems:how redundant information can help. *Annals of Operations Research*, 82:289–308, 1998.

[4] C. Akjiratikarl, P. Yenradee, and P. R. Drake. An improved particle swarm optimization algorithm for care worker scheduling. In *Proceedings of the 7th Asia Pacific industrial engineering and management systems conference*, pages 457–499, 2006.

[5] C. Akjiratikarl, P. Yenradee, and P. R. Drake. PSO-based algorithm for home care worker scheduling in the UK. *Computers & Industrial Engineering*, 53(4):559–583, 2007.

[6] H. Algethami and D. Landa-Silva. A study of genetic operators for the workforce scheduling and routing problem. In *Proceedings of the XI Metaheuristics International Conference (MIC 2015)*, pages 75.1 – 75.11, 2015.

[7] V. D. Angelis. Planning home assistance for AIDS patients in the City of Rome , Italy. *Interfaces*, 28:75–83, 1998.

[8] J. Arroyo and A. Conejo. A parallel repair genetic algorithm to solve the unit commitment problem. *Power Systems, IEEE Transactions on*, 17(4): 1216–1224, Nov 2002.

[9] J. L. Arthur and A. Ravindran. A multiple objective nurse scheduling model. *A I I E Transactions*, 13(1):55–60, 1981.

[10] C. Barnhart, C. A. Hane, E. L. Johnson, and G. Sigismondi. A column generation and partitioning approach for multi-commodity flow problems. *Telecommunication Systems*, 3(3):293–258, 1994.

[11] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

[12] C. Barnhart, C. A. Hane, and P. H. Vance. Using Branch-and-Price-and-Cut to solve origin-destination integer multicommodity flow problems. *Operations Research*, 48(2):318–326, 2000.

[13] D. Barrera, V. Nubia, and A. Ciro-Alberto. A network-based approach to the multi-activity combined timetabling and crew scheduling problem: Workforce scheduling for public health policy implementation. *Computers & Industrial Engineering*, 63(4):802–812, 2012.

[14] M. H. Bassett, J. F. Pekney, and G. V. Reklaitis. Decomposition techniques for the solution of large-scale scheduling problems. *Process Systems Engineering*, 42, 1996.

[15] N. Beaumont. Scheduling staff using mixed integer programming. *European Journal of Operational Research*, 98(3):473–484, 1997.

[16] A. Benchakroun, J. Ferland, and R. Cléroux. Distribution system planning through a generalized benders decomposition approach. *European Journal of Operational Research*, 62(2):149–162, 1992.

[17] J. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962.

[18] I. Berrada, J. A. Ferland, and P. Michelon. A multi-objective approach to nurse scheduling with both hard and soft constraints. *Socio-Economic Planning Sciences*, 30(3):183 – 193, 1996.

[19] A. Billionnet. Integer programming to schedule a hierarchical workforce with variable demands. *European Journal of Operational Research*, 114(1): 105 – 114, 1999.

[20] S. Binato, M. V. F. Pereira, and S. Granville. A new benders decomposition approach to solve power transmission network design problems. *IEEE Transactions on Power Systems*, 16(2):235–240, 2001.

[21] J. Blazewicz, J. Lenstra, and A. Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11 – 24, 1983. ISSN 0166-218X. doi: http://dx.doi.org/10.1016/0166-218X(83)90012-4. URL `http://www.sciencedirect.com/science/article/pii/0166218X83900124`.

[22] V. Borsani, M. Andrea, B. Giacomo, and S. Francesco. A home care scheduling model for human resources. *2006 International Conference on Service Systems and Service Management*, pages 449–454, 2006.

[23] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[24] J. Branke, K. Deb, K. Miettinen, and R. Slowinski. *Multiobjective Optimization: Interactive and Evolutionary Approaches*. Genetic algorithms and evolutionary computation. Springer, 2008. ISBN 9783540889076.

[25] D. Bredstrom and M. Ronnqvist. A branch and price algorithm for the combined vehicle routing and scheduling problem with synchronization constraints. *NHH Dept. of Finance & Management Science Discussion Paper No. 2007/7*, 2007.

[26] D. Bredstrom and M. Ronnqvist. Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. *European Journal of Operational Research*, 191(1):19–31, 2008.

[27] P. Brucker, R. Qu, and E. Burke. Personnel scheduling: Models and complexity. *European Journal of Operational Research*, 210(3):467–473, 2011.

[28] E. K. Burke, P. De Causmaecker, G. V. Berghe, and H. Van Landeghem. The state of the art of nurse rostering. *Journal of Scheduling*, 7(6):441–499, 2004.

[29] E. K. Burke, J. Li, and R. Qu. A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems. *European Journal of Operational Research*, 203(2):484 – 493, 2010.

[30] X. Cai and K. Li. A genetic algorithm for scheduling staff of mixed skills under multi-criteria. *European Journal of Operational Research*, 125(2):359 – 369, 2000.

[31] R. W. Calvo and R. Cordone. A heuristic approach to the overnight security service problem. *Computers & Operations Research*, 30(9):1269 – 1287, 2003.

[32] A. M. Campbell and M. W. Savelsbergh. A decomposition approach for the inventory-routing problem. *Transportation Science*, 38:488–502, 2004.

[33] P. Cappanera and M. G. Scutellá. Joint assignment, scheduling, and routing models to home care optimization: A pattern-based approach. *Transportation Science*, 49:830–852, 2015.

[34] G. Carello and E. Lanzarone. A cardinality-constrained robust model for the assignment problem in home care services. *European Journal of Operational Research*, 236(2):748 – 762, 2014.

[35] J. Castillo-Salazar, D. Landa-Silva, and R. Qu. Workforce scheduling and routing problems: literature survey and computational study. *Annals of Operations Research*, 2014.

[36] J. A. Castillo-Salazar, D. Landa-Silva, and R. Qu. A greedy heuristic for workforce scheduling and routing with time-dependent activities constraints. In *Proceedings of the 4th International Conference on Operations Research and Enterprise Systems (ICORES 2015)*, 2015.

[37] J. Castro-Gutierrez, D. Landa-Silva, and P. J. Moreno. Nature of real-world multi-objective vehicle routing with evolutionary algorithms. *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on*, pages 257–264, 2011.

[38] B. Cheang, H. Li, A. Lim, and B. Rodrigues. Nurse rostering problems – a bibliographic survey. *European Journal of Operational Research*, 151(3): 447–460, 2003.

[39] M. Christiansen. Decomposition of a combined inventory and time constrained ship routing problem. *Transportation Science*, 33, 1999.

[40] N. Christofides, A. Mingozzi, and P. Toth. Exact algorithms for the vehicle routing problem , based on spanning tree and shortest path relaxations. *Mathematical Programming*, 20:255–282, 1981.

[41] P. C. Chu and J. E. Beasley. A genetic algorithm for the generalised assignment problem. *Computers & Operations Research*, 24(1):17–23, 1997.

[42] A. Colorni, M. Dorigo, and V. Maniezzo. Genetic algorithms and highly constrained problems: The time-table case. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature: 1st Workshop, PPSN I Dortmund, FRG, October 1–3, 1990 Proceedings*, pages 55–59. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991.

[43] A. A. Constantino, E. Tozzo, R. L. Pinheiro, D. Landa-Silva, and W. Romao. A variable neighbourhood search for nurse scheduling with balanced preference satisfaction. In *Proceedings of the 17th International Conference on Enterprise Information Systems (ICEIS 2015)*, pages 462–470, 2015.

[44] J.-F. Cordeau, F. Soumis, and J. Desrosiers. A benders decomposition approach for the locomotive and car assignment problem. *Transportation Science*, 34(2):133–149, 2000.

[45] J.-F. Cordeau, G. Stojkovic, F. Soumis, and J. Desrosiers. Benders decomposition for simultaneous aircraft routing and crew scheduling. *Transportation Science*, 35(4):375–388, 2001.

[46] R. Cordone and R. W. Calvo. A heuristic for the vehicle routing problem with time windows. *Journal of Heuristic*, 7:107–129, 2001.

[47] A. M. Costa. A survey on benders decomposition applied to fixed-charge network design problems. *Computers & Operations Research*, 32:1429–1450, 2005.

[48] B. Crevier, J.-F. Cordeau, and G. Laporte. The multi-depot vehicle routing problem with inter-depot routes. *European Journal of Operational Research*, 176(2):756 – 773, 2007.

[49] S. Crimer and W. Zhang. Cut-and-solve: An iterative search strategy for combinatorial optimization problems. *Artificial Intelligence*, 170:714 – 738, 2006.

[50] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960.

[51] R. F. Deckro, E. Winkofsky, J. E. Hebert, and R. Gagnon. A decomposition approach to multi-project scheduling. *European Journal of Operational Research*, 51(1):110 – 118, 1991.

[52] S. Dempe and A. Zemkoho. On the karush–kuhn–tucker reformulation of the bilevel optimization problem. *Nonlinear Analysis: Theory, Methods & Applications*, 75(3):1202 – 1218, 2012.

[53] G. Desaulniers, J. Lavigne, and F. Soumis. Multi-depot vehicle scheduling problems with time windows and waiting costs. *European Journal of Operational Research*, 111(3):479 – 494, 1998.

[54] G. Desaulniers, J. Desrosiers, and M. M. Solomon. Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems. In *Essays and Surveys in Metaheuristics*, volume 15, pages 309–324. Springer US, 2001.

[55] A. Dohn, K. Esben, and C. Jens. The manpower allocation problem with time windows and job-teaming constraints: A branch-and-price approach. *Computers & Operations Research*, 36(4):1145–1157, 2009.

[56] R. Dondo and J. Cerdá. A cluster-based optimization approach for the multi-depot heterogeneous fleet vehicle routing problem with time windows. *European Journal of Operational Research*, 176(3):1478 – 1507, 2007.

[57] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 26(1):29–41, Feb 1996.

[58] K. Dowsland. Nurse scheduling with tabu search and strategic oscillation. *European Journal of Operational Research*, 106(2-3):393–407, 1998.

[59] A. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, 153(1):3–27, 2004.

[60] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD 1996)*, pages 226–231, 1996.

[61] P. Eveborn, M. Rönnqvist, H. Einarsdóttir, M. Eklund, K. Lidén, and M. Almroth. Operations research improves quality and efficiency in home care. *Interfaces*, 39(1):18–34, 2009.

[62] A. Federgruen and P. Zipkin. A combined vehicle routing and inventory allocation problem. *Operations Research*, 32(5):1019–1037, 1984.

[63] A. Field. *Discovering Statistics Using IBM SPSS Statistics*. SAGE Publication Ltd, London, UK, 4 edition, 2013.

[64] M. Firat and C. A. J. Hurkens. An improved mip-based approach for a multi-skill workforce scheduling problem. *Journal of Scheduling*, 15(3): 363–380, 2012.

[65] N. Garg, G. Konjevod, and R. Ravi. A polylogarithmic approximation algorithm for the group steiner problem. *Algorithms*, 37(1):66–84, 2000.

[66] A. M. Geoffrion and G. W. Graves. Multicommodity distribution system design by benders decomposition. *Management Science*, 20(5):822–844, 1974.

[67] F. Glover and C. McMillan. The general employee scheduling problem. an integration of {MS} and {AI}. *Computers & Operations Research*, 13(5): 563 – 573, 1986.

[68] D. Goldberg. *Genetic Algorithms*. Pearson Education, 2006.

[69] M. Grötschel, M. Jünger, and G. Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32(6):1195–1220, 1984.

[70] C. Heimerl and K. Rainer. Scheduling and staffing multiple projects with a multi-skilled workforce. *OR Spectrum*, 32(2):343–368, 2009.

[71] W. Herroelen, B. D. Reyck, and E. Demeulemeester. Resource-constrained project scheduling: A survey of recent developments. *Computers & Operations Research*, 25(4):279 – 302, 1998. ISSN 0305-0548. doi: http://dx.doi.org/10.1016/S0305-0548(97)00055-5. URL `http://www.sciencedirect.com/science/article/pii/S0305054897000555`.

[72] G. Hiermann, M. Prandtstetter, A. Rendl, J. Puchinger, and G. R. Raidl. Metaheuristics for solving a multimodal home-healthcare scheduling problem. *Central European Journal of Operations Research*, 23:89–113, 2015.

[73] J. J. E. Kelley. The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, 8(4):703–712, 1960.

[74] A. Jan, M. Yamamoto, and A. Ohuchi. Evolutionary algorithms for nurse scheduling problem. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1, pages 196–203, 2000.

[75] B. Jaumard, F. Semet, and T. Vovor. A generalized linear programming model for nurse scheduling. *European Journal of Operational Research*, 107 (1):1 – 18, 1998.

[76] Y. Kergosien, C. Lenté, and J.-C. Billaut. Home health care problem, an extended multiple travelling salesman problem. In *Proceedings of the 4th multidisciplinary international scheduling conference: Theory and applications (MISTA 2009), Dublin, Ireland*, pages 85–92, 2009.

[77] C. Koulamas, S. Antony, and R. Jaen. A survey of simulated annealing applications to operations research problems. *Omega*, 22(1):41 – 56, 1994.

[78] D. Landa-Silva, Y. Wang, P. Donovan, G. Kendall, and S. Way. Hybrid heuristic for multi-carrier transportation plans. In *The 9th Metaheuristics International Conference (MIC 2011)*, pages 221–229, 2011.

[79] G. Laporte. The vehicle routing problem : An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59:345–358, 1992.

[80] J. K. Lenstra and A. Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227, 1981.

[81] H. Li and W. Keith. Scheduling projects with multi-skilled personnel by a hybrid MILP/CP benders decomposition algorithm. *Journal of Scheduling*, 12(3):281–298, 2008.

[82] Y. Li, L. Andrew, and R. Brian. Manpower allocation with time windows and job-teaming constraints. *Naval Research Logistics*, 52(4):302–311, 2005.

[83] L. Lian and E. Castelain. A decomposition-based heuristic approach to solve general delivery problem. In *Proceedings of the World Congress on Engineering and Computer Science*, volume 2, 2009.

[84] L. Liberti. An exact reformulation algorithm for large nonconvex nlps involving bilinear terms. *Global Optimization*, 36(2):161–189, 2006.

[85] L. Liberti. Reformulations in mathematical programming: automatic symmetry detection and exploitation. *Mathematical Programming*, 131(1–2):273–304, 2012.

[86] L. Liberti and C. C. Pantelides. Reformulations in mathematical programming: Definitions and systematics. *RAIRO - Operations Research*, 43:55–85, 1 2009.

[87] L. Liberti, S. Cafieri, and F. Tarissan. Reformulations in mathematical programming: A computational approach. In A. Abraham, A.-E. Hassanien, P. Siarry, and A. Engelbrecht, editors, *Foundations of Computational Intelligence*, volume 3 of *Studies in Computational Intelligence*, pages 153–234. Springer Berlin Heidelberg, 2009.

[88] A. Lim and F. Wang. Multi-depot vehicle routing problem: a one-stage approach. *Automation Science and Engineering, IEEE Transactions on*, 2(4): 397–402, 2005.

[89] C. Lim. Relationship among benders, dantzig–wolfe, and lagrangian optimization. In J. J. Cochran, L. A. Cox, P. Keskinocak, J. P. Kharoufeh, and J. C. Smith, editors, *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc., 2010.

[90] L. A. Lorena and E. L. Senne. A column generation approach to capacitated p-median problems. *Computers & Operations Research*, 31(6):863 – 876, 2004.

[91] D. S. Mankowska, F. Meisel, and C. Bierwirth. The home health care routing and scheduling problem with interdependent services. *Health Care Management Science*, 17:15–30, 2014.

[92] A. Mercier and F. Soumis. An integrated aircraft routing, crew scheduling and flight retiming model. *Computers & Operations Research*, 34(8):2251 – 2265, 2007.

[93] A. Mercier, J.-F. Cordeau, and F. Soumis. A computational study of Benders decomposition for the integrated aircraft routing and crew scheduling problem. *Computers & Operations Research*, 32(6):1451 – 1476, 2005.

[94] M. Misir, P. Smet, K. Verbeeck, and G. Vanden Berghe. Security personnel routing and rostering: a hyper-heuristic approach. In *Proceedings of the*

3rd International Conference on Applied Operational Research, ICAOR2011, Istanbul, Turkey, pages 193–205, August 2011.

[95] L. Mockus and G. Reklaitis. Mathematical programming formulation for scheduling of batch operations based on nonuniform time discretization. *Computers & Chemical Engineering*, 21(10):1147 – 1156, 1997.

[96] J. M. Mulvey and M. P. Beck. Solving capacitated clustering problems. *European Journal of Operational Research*, 18(9):339–348, 1984.

[97] P. Munari and J. Gondzio. Column generation and branch-and-price with interior point methods. In *Proceeding Series of the Brazilian Society of Applied and Computational Mathematics*, volume 3, 2015.

[98] B. Murovec and P. Šuhel. A repairing technique for the local search of the job-shop problem. *European Journal of Operational Research*, 153(1):220 – 238, 2004.

[99] G. L. Nemhauser and L. A. Wolsey. *Integer and combinatorial optimization*. Discrete Mathematics and Optimization. Wiley, 1988.

[100] I. H. Osman and N. Christofides. Capacitated clustering problems by hybrid simulated annealing and tabu search. *International Transactions in Operational Research*, 1(3):317 – 336, 1994.

[101] K. Papoutsis, C. Valouxis, and E. Housos. A column generation approach for the timetabling problem of Greek high schools. *Journal of the Operational Research Society*, 54:230–238, 2003.

[102] H.-S. Park and C.-H. Jun. A simple and fast algorithm for k-medoids clustering. *Expert Systems with Applications*, 36(2, Part 2):3336 – 3341, 2009.

[103] R. L. Pinheiro, D. Landa-Silva, and J. Atkin. A variable neighbourhood search for the workforce scheduling and routing problem. *Advances in Nature and Biologically Inspired Computing, Series Advances in Intelligent Systems and Computing*, pages 247–259, 2015.

[104] W. B. Powell. *Approximate Dynamic Programming*. Wiley, 2011.

[105] J. Puchinger and G. R. Raidl. An evolutionary algorithm for column generation in integer programming: An effective approach for 2d bin packing. In X. Yao, E. K. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervós,

J. A. Bullinaria, J. E. Rowe, P. Tiňo, A. Kabán, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VIII: 8th International Conference, Birmingham, UK, September 18-22, 2004. Proceedings*, pages 642–651. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[106] C. D. Randazzo, H. P. L. Luna, and P. Mahey. Benders decomposition for local access network design with two technologies. *Discrete Mathematics & Theoretical Computer Science*, 4(2), 2001.

[107] M. S. Rasmussen, T. Justesen, A. Dohn, and J. Larsen. The home care crew scheduling problem: Preference-based visit clustering and temporal dependencies. *European Journal of Operational Research*, 219(3):598–610, 2012.

[108] M. Reimann, K. Doerner, and R. F. Hartl. D-Ants: Savings based ants divide and conquer the vehicle routing problem. *Computers & Operations Research*, 31(4):563 – 591, 2004.

[109] T. J. V. Roy and L. A. Wolsey. Solving mixed integer programming problems using automatic reformulation. *Operations Research*, 35(1):45–57, 1987.

[110] N. Sahinidis and I. Grossmann. Reformulation of multiperiod MILP models for planning and scheduling of chemical processes. *Computers & Chemical Engineering*, 15(4):255 – 272, 1991.

[111] M. Salani and I. Vacca. Branch and price for the vehicle routing problem with discrete split deliveries and time windows. *European Journal of Operational Research*, 213:470–477, 2011.

[112] M. Savelsbergh. A branch-and-price algorithm for the generalized assignment problem. *Operations Research*, 45(6):831–841, 1997.

[113] M. W. P. Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research*, 4(1):285–305, 1985.

[114] S. U. Seçkiner, H. Gökçen, and M. Kurt. An integer programming model for hierarchical workforce scheduling problem. *European Journal of Operational Research*, 183(2):694 – 699, 2007.

[115] H. D. Sherali and A. Alameddine. A new reformulation-linearization technique for bilinear programming problems. *Global Optimization*, 2(4): 379–410, 1992.

[116] H. D. Sherali and C. H. Tuncbilek. A reformulation-convexification approach for solving nonconvex quadratic programming problems. *Global Optimization*, 7(1):1–31, 1995.

[117] H. D. Sherali and C. H. Tuncbilek. New reformulation linearization/convexification relaxations for univariate and multivariate polynomial programming problems. *Operations Research Letters*, 21(1):1 – 9, 1997.

[118] M. M. Solomon. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research*, 35(2), 1987.

[119] É. Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23(8):661–673, 1993.

[120] É. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31(2):170–186, 1997.

[121] A. Trautsamwieser and P. Hirsch. Optimization of daily scheduling for home health care services. *Journal of Applied Operational Research*, 3:124–136, 2011.

[122] V. Valls, Ángeles Pérez, and S. Quintanilla. Skilled workforce scheduling in service centres. *European Journal of Operational Research*, 193(3):791 – 804, 2009.

[123] F. Vanderbeck. On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. *Operations Research*, 48(1):111–128, 2000.

[124] F. Vanderbeck and L. A. Wolsey. An exact algorithm for IP column generation. *Operations Research Letters*, 19(4):151 – 159, 1996.

[125] F. Vanderbeck and L. A. Wolsey. Reformulation and decomposition of integer programs. In M. Junger et al., editors, *50 Years of Integer Programming 1958-2008*, pages 431–502. Springer Berlin Heidelberg, 2010.

[126] A. Vela, S. Solak, W. Shinghose, and J.-P. Clark. A mixed integer program for flight-level assignment and speed control for conflict resolution. In *Proceeding in Joint 48th IEEE Conference on Decision and Control and 28th Chinese Control Conference*, 2009.

[127] D. M. Warner. Scheduling nursing personnel according to nursing preference: A mathematical programming approach. *Operations Research*, 24 (5):842–856, 1976.

[128] T.-H. Wu, C. Low, and J.-W. Bai. Heuristic solutions to multi-depot locaiton-routing problems. *Computers & Operations Research*, 29:1393–1415, 2002.

[129] K. Ziarati, F. Soumis, J. Desrosiers, S. Gélinas, and A. Saintonge. Locomotive assignment with heterogeneous consists at CN North America. *European Journal of Operational Research*, 97(2):281 – 292, 1997.

[130] M. Zweben, M. Deale, and R. Gargan. Anytime rescheduling. In *Proceeding of DARPA Workshop Innovative Approaches to Planning and Scheduling*, 1990.

[131] M. Zweben, E. Davis, B. Daun, and M. Deale. Scheduling and rescheduling with iterative repair. *Systems, Man and Cybernetics, IEEE Transactions on*, 23(6):1588–1596, Nov 1993.

Appendices

Appendix A

Models in OPL

A.1 MIP Model for HHC Problem in OPL

```
/*********************************************
 * OPL 12.4 Model
 * Author: wxl
 *********************************************/

/********* Set and Parameter declairation ***********/
{string} clientID = ...;  // set of client
{string} depID = ...;  // set of Depot
{string} arrID = ...;  // set of Finishing Place
{string} locationID = depID union clientID union arrID;
{string} depLocationID = depID union clientID;
{string} arrLocationID = arrID union clientID;

tuple Availability{
  int workerID;
  int aFrom;
  int aTo;
}

{Availability} available = ...;

tuple Worker{
  int workerID;
  int workerName;
  float workLimit;
  string workerGender;
  string depID;
  string arrID;
}

{Worker} workers = ...;   //List of workers

//Demand data structure
tuple Demand{
  string demandID;
  string locationID;
  float readyTime;
  float duedate;
  float clientDemand;
  float serviceTime;
};

{Demand} demands = ...;  //List of visit
{Demand} depotLocation = ...;  //Departure location
{Demand} arrivalLocation = ...;  //Arrival location
{Demand} departure = depotLocation union demands; //Node that have leaving edges
{Demand} arrival = arrivalLocation union demands; //NOde that have entering edges
{Demand} location = depotLocation union demands union arrivalLocation; //All node


float Distances[depLocationID,arrLocationID] = ...; // Distances matrix
float Times[depLocationID,arrLocationID] = ...; // Times matrix
float Preferences[workers,demands]= ...; //preference of workers
float WorkerLocation[workers,demands] = ...;//worker location restriction
int M = 100000;

float areaPenalty = ...;
float timePenalty = ...;
float unAssignPenalty = ...;
float unFulfillPenalty = ...;
float travellingWeight = ...;

/*****Variable declaration **********/
dvar boolean assign[workers,departure,arrival]; // decision variable for choosing route
dvar float+ timeArr[workers,location]; // departure time of each car
dvar int+ dummy[demands];
dvar boolean extraArea[arrival,workers];
dvar boolean isOvertime[arrival,workers];

// calculate the total distances of vehicle
dexpr float TotalDistance = sum(c in workers,i in departure, j in arrival)
assign[c,i,j]*Distances[i.locationID,j.locationID];

// Worker region soft violation
dexpr float TotalAreaViolation = sum(c in workers,i in arrival)extraArea[i,c];

// Worker time soft violation
dexpr float TotalTimeViolation = sum(c in workers,i in arrival)isOvertime[i,c];

// Preferences
dexpr float TotalPreferences = sum(c in workers, d in demands,i in departure)
                                    assign[c,i,d]*(Preferences[c,d]-1);

// Number of unassigned visit
dexpr float TotalDummy = sum(a in demands)(dummy[a]);

/************** Objective function **************/
minimize  travellingWeight*TotalDistance
+ areaPenalty*TotalAreaViolation
+ timePenalty*TotalTimeViolation
+TotalPreferences
+ unAssignPenalty*TotalDummy ;
```

```
/*************** Constraints ********************/
subject to {

// Constraint: Visit Assignment Constraint
CustomerMustBeVisit:  forall(j in demands)
    sum(c in workers,i in departure)(assign[c,i,j]) + dummy[j] == j.clientDemand;

// Constraint: Route Continuity Constraint
BalanceFlow:  forall(c in workers,j in demands)
    sum(i in departure)assign[c,i,j] == sum(i in arrival) assign[c,j,i];
UniqueArrival:  forall(c in workers,j in arrivalLocation )
    sum (i in departure)assign[c,i,j] <= 1;
UniqueDeparture:  forall(c in workers,i in depotLocation)
    sum (j in arrival)assign[c,i,j] <= 1;

// Constraint: Start Location Constraint
StrictDeparture:  forall(c in workers)
    sum(i in depotLocation, j in demands :c.depID != i.locationID) assign[c,i,j] ==0;

// Constraint: End Location Constraint
StrictReturn: forall(c in workers)
    sum(i in arrivalLocation, j in demands: c.arrID != i.locationID) assign[c,j,i] == 0;

// Constraint: Travel Time Feasibility Constraint
JobSequence:  forall(c in workers, i in departure,j in demands )
    timeArr[c,j]+M*(1-assign[c,i,j]) >= timeArr[c,i]+i.serviceTime
                   + Times[i.locationID,j.locationID]*assign[c,i,j] ;
JobFinishing: forall(c in workers, i in demands, j in arrivalLocation)
    timeArr[c,j]+M*(1-assign[c,i,j]) >= timeArr[c,i]+i.serviceTime
                   + Times[i.locationID,j.locationID]*assign[c,i,j];

// Constraint: Time Window Constraint
LowerTimeWindow:  forall(c in workers, i in demands)
    timeArr[c,i]>= i.readyTime*sum(j in departure)assign[c,j,i];
UpperTimeWindow:  forall(c in workers, i in demands)
    timeArr[c,i]<= i.duedate*sum(j in departure)assign[c,j,i];

//Constraint: Skill and Qualification Constraint
Qualification: forall(c in workers, d in demands, i in departure)
assign[c,i,d] <= Preferences[c,d];

// Constraint: Working Hour Limit Constraint
CapConst:  forall(c in workers)
    sum(i in demands,j in arrival)(assign[c,i,j]*i.serviceTime) <= c.workLimit;

//Constraint: Workforce Time Availability Constraint
AvailableLowCons: forall(c in workers,j in departure, i in demands,
    a in available:a.workerID == c.workerID)
        a.aFrom - timeArr[c,i] <= (1-assign[c,j,i] + isOvertime[i,c])*M;
AvailableUpCons: forall(c in workers,j in departure, i in demands,
    a in available:a.workerID == c.workerID)
        timeArr[c,i] + i.serviceTime - a.aTo  <= (1 - assign[c,j,i] + isOvertime[i,c])*M;

// Constraint: Workforce Region Availability Constraint
AreaRestrict: forall(c in workers, j in demands )
```

```
    sum(i in departure) assign[c,i,j]-extraArea[j,c] <= WorkerLocation[c,j];

// Constraint: Valid Equality, Route from start to finish is 0
PreventNoTask: forall(c in workers)
    sum(i in depotLocation, j in arrivalLocation)assign[c,i,j] == 0;

// Constraint: Valid Inequality, Worker will not make revisit
NoCirculation:  forall(c in workers,i in demands,
    j in demands:i.locationID!=j.locationID )
        assign[c,i,j] + assign[c,j,i] <=1;
MustChangeToOtherVisit:  forall(c in workers,i in demands)assign[c,i,i] == 0;
}
/*************** End Model ********************/
```

A.2 MIP Model for WSRP in OPL

```
/*********************************************
 * OPL 12.4 Model
 * Author: wxl
 *********************************************/

/********* Set and Parameter declaration ***********/
{string} locationID = ...;  // set of client

// Workforce Availability
tuple Availability{
  int workerID;
  int aFrom;
  int aTo;
}

{Availability} available = ...;

//worker data structure
tuple Worker{
  int workerID;
  float workLimit;
  string depID;
  string arrID;
}

{Worker} workers = ...;   //List of workers

//Demand data structure
tuple Demand{
  int demandID;
  string locationID;
  float readyTime;
  float duedate;
  float clientDemand;
  float serviceTime;
  int priority;
};

//Time dependent activities
```

```
tuple spCond{
int d1ID;   //from activity
int d2ID;   //to activity
float time; //time different
};

{spCond} precLeq = ...; // list of visit that d1ID <= d2ID
{spCond} precGeq = ...; // list of visit that d1ID >= d2ID
{spCond} Overlap = ...;// list of visit that d1ID and d2ID are time overlap
{int} forceAssign = ...;

{Demand} demands = ...;  //List of services
{Demand} depotLocation = ...;  //For modelling reason
{Demand} arrivalLocation = ...;  //For modelling reason
{Demand} departure = depotLocation union demands; //Places that workers departure from
{Demand} arrival = arrivalLocation union demands; //Places that workers arrival to
{Demand} location = depotLocation union demands union arrivalLocation; //All services

// Matrices
float Distances[locationID,locationID] = ...; // Distances matrix
float Times[locationID,locationID] = ...; // Times matrix
float Preferences[workers,demands]= ...; //preference of workers
int Compatibility[workers,demands]=...; //compatibility of workers
int M = 100000;

float unAssignPenalty = ...;
float travellingWeight = ...;
float timeWeight =...;
float preferenceWeight = ...;

/*****Variable declaration **********/
dvar boolean assign[workers,departure,arrival]; // decision variable for choosing route
dvar float+ timeArr[workers,location]; // departure time of each car
dvar boolean dummy[demands];

dexpr float TotalDistance = sum(c in workers,i in departure, j in arrival)
    assign[c,i,j]*Distances[i.locationID,j.locationID];

// calculate the total arrival time
dexpr float TotalTravelTime = sum(c in workers,i in departure, j in arrival)
    assign[c,i,j]*Times[i.locationID,j.locationID];

// calculate the total worker's preferences
dexpr float TotalPreferences = sum(c in workers, d in demands,i in departure)
    assign[c,i,d]*(Preferences[c,d]);

dexpr float TotalDummy = sum(a in demands)(dummy[a]*a.priority);


/************** Objective function **************/
minimize travellingWeight*TotalDistance
+ timeWeight*TotalTravelTime
+ preferenceWeight*TotalPreferences
+ unAssignPenalty*TotalDummy
;
```

```
/*************** Constraints ********************/
subject to {

// Constraint: Visit Assignment Constraint
CustomerMustBeVisit:  forall(j in demands)
    sum(c in workers,i in departure)(assign[c,i,j]) + dummy[j] >= 1;
FillDemand:  forall(j in demands)
    sum(c in workers,i in departure)(assign[c,i,j]) +dummy[j]*j.clientDemand == j.clientDemand;
OneWorker: forall(j in demands, c in workers:Compatibility[c][j]==1)
    sum(i in departure)(assign[c,i,j]) <= 1;

// Constraint: Route Continuity Constraint
BalanceFlow:  forall(c in workers,j in demands:Compatibility[c][j]==1)
    sum(i in departure)assign[c,i,j] == sum(i in arrival) assign[c,j,i];
UniqueArrival:  forall(c in workers,j in arrivalLocation )
    sum (i in departure)assign[c,i,j] <= 1;
UniqueDeparture:  forall(c in workers,i in depotLocation)
    sum (j in arrival)assign[c,i,j] <= 1;

// Constraint: Start Location Constraint
StrictDeparture:  forall(c in workers)
    sum(i in depotLocation, j in demands :c.depID != i.locationID) assign[c,i,j] ==0;

// Constraint: End Location Constraint
StrictReturn: forall(c in workers)
    sum(i in arrivalLocation, j in demands: c.arrID != i.locationID) assign[c,j,i] == 0;

// Constraint: Travel Time Feasibility Constraint
JobSequence:  forall(c in workers, i in departure,j in demands:Compatibility[c][j]==1)
    timeArr[c,j]+M*(1-assign[c,i,j]) >= timeArr[c,i]+i.serviceTime
        + Times[i.locationID,j.locationID]*assign[c,i,j] ;
JobFinishing: forall(c in workers, i in demands, j in arrivalLocation:Compatibility[c][i]==1)
    timeArr[c,j]+M*(1-assign[c,i,j]) >= timeArr[c,i]+i.serviceTime
        + Times[i.locationID,j.locationID]*assign[c,i,j];

// Constraint: Time Window Constraint
LowerTimeWindow:  forall(c in workers, i in demands:Compatibility[c][i]==1)
    timeArr[c,i]>= i.readyTime*sum(j in departure)assign[c,j,i];
UpperTimeWindow:  forall(c in workers, i in demands:Compatibility[c][i]==1)
    timeArr[c,i]<= i.duedate*sum(j in departure)assign[c,j,i];

// Constraint: Skill and Qualification Constraint
CompatCons: sum(c in workers, i in departure, j in demands: Compatibility[c][j]==0)
    assign[c,i,j] <= 0;

// Constraint: Time Availability Constraint
AvailableLowCons: forall(c in workers,j in departure, i in demands,
    a in available:a.workerID == c.workerID)
        a.aFrom +Times[j.locationID,i.locationID] <= (1-assign[c,j,i])*M +  timeArr[c,i];
AvailableUpCons: forall(c in workers,j in demands, i in arrival,
    a in available:a.workerID == c.workerID)
        timeArr[c,i]  <= a.aTo;

// Constraint: Synchronisation Constraint
TaskTimeLeq: forall(i in demands, c1 in workers, c2 in workers:c1!=c2)
    timeArr[c1,i]-M*(2-sum(j in departure)(assign[c1,j,i]+assign[c2,j,i])) <= timeArr[c2,i];
TaskTimeGeq: forall(i in demands, c1 in workers, c2 in workers:c1!=c2)
    timeArr[c1,i]+M*(2-sum(j in departure)(assign[c1,j,i]+assign[c2,j,i])) >= timeArr[c2,i];

// Constraint: Min, Max, Min-Max Constraint
spConsLeq: forall(s in precLeq, i in demands, j in demands,
    c1 in workers,c2 in workers:s.d1ID==i.demandID && s.d2ID==j.demandID)
        timeArr[c2,j] <= timeArr[c1,i]+s.time +(2 - sum(k in departure)assign[c1,k,i]
            - sum(k in departure)assign[c2,k,j])*M;
spConsGeq: forall(s in precGeq, i in demands, j in demands,
    c1 in workers,c2 in workers:s.d1ID==i.demandID && s.d2ID==j.demandID)
        timeArr[c2,j] +(2 - sum(k in departure)assign[c1,k,i]
            - sum(k in departure)assign[c2,k,j])*M >= timeArr[c1,i]+s.time;

// Constraint: Overlap Constraint
spConsOverlap1: forall(s in Overlap, i in demands, j in demands,
    c1 in workers,c2 in workers:s.d1ID==i.demandID && s.d2ID==j.demandID)
        timeArr[c2,j] -(2 - sum(k in departure)assign[c1,k,i]
            - sum(k in departure)assign[c2,k,j])*M <= timeArr[c1,i]+i.serviceTime;
spConsOverlap2: forall(s in Overlap, i in demands, j in demands,
    c1 in workers,c2 in workers:s.d1ID==i.demandID && s.d2ID==j.demandID)
        timeArr[c2,j]+j.serviceTime +(2 - sum(k in departure)assign[c1,k,i]
            - sum(k in departure)assign[c2,k,j])*M >= timeArr[c1,i];

// Constraint: Valid Equality, Route from start to finish is 0
PreventNoTask: forall(c in workers)
    sum(i in depotLocation, j in arrivalLocation)assign[c,i,j] == 0;


// Constraint: Valid Inequality, Worker will not make revisit
NoCustomerCirculation:  forall(c in workers,i in demands,j in demands
    :i.locationID!=j.locationID&&Compatibility[c][j]==1&&Compatibility[c][i]==1)
        assign[c,i,j] + assign[c,j,i] <=1;
TaskMustChange:  forall(c in workers,i in demands:Compatibility[c][i]==1)
    assign[c,i,i] == 0;
}
/*************** End Model *******************/
```

A.3 Compact MIP Model for HHC Problem in OPL

```
/*********************************************
 * OPL 12.4 Model
 * Author: wxl
 *********************************************/

 /********* Set and Parameter declairation ***********/
{int} workerID = ...; // List of Worker ID

// Structure of Visits
tuple Demand{
  int demandID;
  int clientDemand; // Number of worker required
  float duration; // Duration of visits
};
{Demand} task = ...;
```

```
float Cost[workerID,task] = ...; // Cost assign worker to visit
int Conflict[task,task] = ...; // Conflict matrix, 1 means conflict between pair
int Compat[workerID,task] = ...; // Worker is qualified to make visit
int HourLimit[workerID] = ...; // Workforce maximum working hours
float M =...; // Unassigned cost

/*****Variable declaration **********/
dvar boolean assign[workerID,task];
dvar int+ unassigned[task];

dexpr float AssignCost = sum(w in workerID, t in task)assign[w,t]*Cost[w,t];
dexpr float unassignedCost = sum(t in task)M*unassigned[t];

/************** Objective function **************/
minimize AssignCost + unassignedCost;

/*************** Constraints ********************/
subject to {
// Constraint: Visit Assignment Constraint
assigningCons: forall(t in task)
    sum(w in workerID)assign[w,t]+unassigned[t] == t.clientDemand;

// Constraint: Working Hour Limit Constraint
CapConst:  forall(w in workerID)
    sum(t in task)(assign[w,i]*i.duration) <= HourLimit[c];

// Constraint: Conflict Avoidance Constraint
conflictCons: forall(t1 in task, t2 in task, w in workerID
    : t1!=t2 && Conflict[t1,t2]!=0 && Compat[w,t1]!=0 && Compat[w,t2]!=0)
        (assign[w,t1]+assign[w,t2])*Conflict[t1,t2] <= 1;
}
/*************** End Model ********************/
```

Appendix B

Number of visits and number of workers by regions

Table B.1: Number of visits (V) and number of workers (K) by regions of instance set A.

Region	A-01		A-02		A-03		A-04		A-05		A-06		A-07	
	V	K	V	K	V	K	V	K	V	K	V	K	V	K
1	1	9	1	9	1	9	0	6	0	6	0	8	0	8
2	10	18	8	17	10	17	9	14	4	14	7	16	4	16
3	1	9	7	22	1	9	5	19	1	19	1	8	2	21
4	1	9	15	20	6	22	14	17	8	17	1	8	7	19
5	3	23			20	20					1	8		
6	15	21									1	8		
7											5	21		
8											12	19		

Table B.2: Number of visits (V) and number of workers (K) by regions of instance set B.

Region	B-01		B-02		B-03		B-04		B-05		B-06		B-07	
	V	K	V	K	V	K	V	K	V	K	V	K	V	K
1	1	10	0	10	0	20	0	20	1	19	0	20	1	20
2	3	25	2	25	6	34	3	34	4	32	3	32	5	32
3	3	25	5	25	5	34	4	34	3	32	3	32	2	32
4	12	25	1	25	15	34	16	34	12	32	12	32	11	32
5	8	25	4	25	6	34	7	34	6	32	6	32	6	32
6	9	25			30	34			22	32	21	32	25	32
7					7	34			1	32	1	32	11	32
8									12	32	11	32		

Table B.3: Number of visits (V) and number of workers (K) by regions of instance set C.

Region	C-01		C-02		C-03		C-04		C-05		C-06		C-07	
	V	K	V	K	V	K	V	K	V	K	V	K	V	K
1	95	1	1	0	91	3	12	2	12	2	9	62	0	2
2	15	390	3	131	5	406	3	403	1	275	2	4	1	111
3	3	132	1	8	1	139	1	138	1	113	70	88	1	8
4	2	18	2	84	10	370	5	284	5	283	3	181	1	2
5	16	333			29	134	5	126	4	27	1	10	1	4
6	36	133			1	112	1	101	6	114	5	7	2	34
7	1	109			13	124	3	32			13	397		
8	9	125					2	116			36	83		
9											1	89		
10											2	17		
11											16	76		

Table B.4: Number of visits (V) and number of workers (K) by regions of instance set D.

Region	D-01		D-02		D-03		D-04		D-05		D-06		D-07	
	V	K	V	K	V	K	V	K	V	K	V	K	V	K
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	48	18	41	19	72	22	68	22	56	22	78	20	77	20
3	23	4	19	4	25	5	23	5	24	5	23	5	23	5
4	47	28	2	28	73	27	67	27	71	26	78	25	72	25
5	98	27	1	27	106	28	95	28	103	31	113	32	112	32
6	19	17	47	28	20	16	10	16	21	13	26	12	26	12
7	76	57	3	19	105	62	84	62	100	61	109	66	102	65
8	29	13	85	27	30	14	30	14	25	15	37	14	34	14
9	1	57	151	58	3	62	3	62	2	61	3	66	2	65
10	4	28	11	17	1	13	1	13	1	11	1	11	1	11
11	2	27	71	58	3	14	1	14	1	15	2	14	1	14
12	3	18	22	13	5	27	4	27	7	26	8	25	7	25
13	132	57			7	27	1	27	3	30	5	32	5	32
14					5	20	3	20	5	21	5	20	5	20
15					129	62	130	62	119	61	122	65	143	65

Table B.5: Number of visits (V) and number of workers (K) by regions of instance set E.

Region	E-01		E-02		E-03		E-04		E-05		E-06		E-07	
	V	K	V	K	V	K	V	K	V	K	V	K	V	K
1	0	26	0	26	0	15	0	14	0	12	0	12	0	8
2	77	173	79	174	89	188	56	187	92	196	47	196	88	208
3	180	182	178	183	218	197	187	196	197	205	163	205	249	217
4	48	151	47	152	43	169	39	168	50	177	33	177	47	189
5	52	149	59	150	42	168	46	167	52	176	41	176	46	189
6	5	68	7	68	6	59	3	58	4	59	2	59	12	59
7	2	105	1	103	3	91	2	53	3	53	2	53	1	89
8	22	62	4	105	1	94	6	53	18	53	1	53	1	94
9	2	60	16	62	4	54	2	51	5	51	1	51	3	53
10	5	77	1	60	17	54	3	69	9	69	2	72	18	53
11	17	80	7	77	1	52	2	72	8	72	3	12	1	51
12	4	58	15	80	8	70	2	52	8	50	2	42	8	72
13	4	26	7	58	12	73	3	14	7	12	4	42	12	75
14			4	26	12	53			2	42			6	49
15					6	15			6	42			5	8
16													1	45

Table B.6: Number of visits (V) and number of workers (K) by regions of instance set F.

Region	F-01		F-02		F-03		F-04		F-05		F-06		F-07	
	V	K	V	K	V	K	V	K	V	K	V	K	V	K
1	2	0	5	0	5	0	0	0	0	0	0	0	4	0
2	8	0	9	0	4	0	4	0	1	0	1	64	1	63
3	1	31	2	52	2	1	1	63	1	61	2	38	3	39
4	135	17	2	30	4	81	7	74	1	35	4	67	4	0
5	4	54	5	64	245	33	270	33	1	37	260	37	5	80
6	31	13	135	18	52	14	52	15	5	57	58	15	232	36
7	2	55	5	65	1	83	3	81	256	37	9	86	61	17
8	34	26	30	13	49	41	40	41	59	15	34	41	10	92
9	1	1	35	26	2	1	108	23	11	83	136	27	50	43
10	47	18	1	1	119	23	1	6	33	41	1	6	2	1
11	2	7	41	18	1	4	85	1	127	27	2	7	158	24
12	1	4	2	13	4	3	2	8	1	6	85	1	2	6
13	117	3	1	4	95	1	1	8	3	6	37	29	3	2
14	4	10	1	3	4	10	34	28	1	4	11	12	90	0
15	60	24	121	3	39	27	4	7	2	2	10	1	3	10
16	1	1	8	10	1	0	13	1	91	1	1	4	1	11
17	3	33	62	23	6	9	4	3	40	29	1	1	45	35
18	3	4	1	0	13	1	1	1	1	0	1	1	1	0
19	1	3	6	33	1	4	1	20	12	12	1	20	16	10
20	1	5	3	5	1	5	3	9	10	1	2	10	1	2
21	4	34	4	3	6	5	2	0	2	8	120	31	15	1
22	1	0	4	33	2	3	3	29	1	6	20	20	12	23
23	17	11	1	0	2	2	19	19	2	4	51	25	1	6
24	40	15	2	0	1	2	53	24	1	1	4	11	6	4
25	1	0	20	11	1	0	2	10	1	3	11	10	1	1
26	34	13	38	15	1	19	1	0	1	1	4	9	1	1
27	8	70	1	0	4	9	11	9	1	1	2	9	1	3
28	13	12	37	13	4	29	1	8	1	0	29	10	1	1
29	3	4	7	70	1	0	37	9	1	1	21	40	1	1
30	1	41	14	12	24	18	19	35	1	20	16	10	1	7
31	68	7	1	4	50	23	12	9	3	10	1	1	2	3
32	4	24	1	40	1	10	1	1	134	31	8	22	1	19
33	1	2	69	7	1	0	68	10	2	2	50	10	6	13
34	27	7	22	64	11	9	3	50	1	0	4	29	148	26
35	0	4	31	7	2	8	3	0	1	0	4	0	3	0
36	15	3	2	4	38	9	6	3	21	20	72	109	1	0
37	1	98	14	3	13	34	0	1	54	25	135	40	22	22
38	23	110	1	108	12	9	4	1	6	11	18	3	60	29
39	91	39	2	1	3	0	60	106	13	10	119	31	1	12
40	20	10	99	40	67	9	132	38	7	9	1	0	13	16
41	96	21	20	9	13	63	1	3	3	9	8	19	12	13
42	12	22	92	22	3	0	23	5	23	40	5	16	6	11
43	1	20	13	21	26	8	124	28	34	10	210	51	2	11
44	249	60	2	19	2	5	7	19	15	10	13	45	40	11
45	23	54	248	61	3	4	2	16	6	21			7	36
46			23	58	5	3	206	51	54	10			16	11
47					139	38	14	46	1	2			2	1
48					1	2			1	20			9	19
49					27	5			3	0			90	11
50					121	30			10	7			9	62
51					15	19			0	5			1	1
52					1	16			1	4			3	0
53					214	52			152	40			42	7
54					17	48			21	4			5	5
55									121	31			1	0
56									4	19			4	3
57									1	16			4	4
58									224	50			130	36
59									15	44			14	3
60													99	34
61													8	19
62													5	16
63													209	48
64													19	43

Appendix C

WSRP with Time-dependent Activities Constraints instances

Table C.1: Summary of instances used in Chapter 6.

Instance	# Workers	# Visits	VisitDur	Worker/Visit	WorkerHour	VistWindow	Horizon	#TimeDep					
								Sync	Overlap	Min	Max	MinMax	Total
Security Guard Patrolling Instances (Sec)													
10_District0_WSRPv2	13	52	282.1154	1.3077	24	415.3846	1440	2	0	3	0	2	7
10_District1_WSRPv2	29	118	531.4831	1.178	24	501.4746	1440	3	5	4	4	6	22
10_District2_WSRPv2	13	52	368.9423	1.0962	24	452.7115	1440	1	1	0	2	2	6
10_District3_WSRPv2	14	58	384.569	1.2759	24	405	1440	1	1	2	2	2	8
10_District4_WSRPv2	39	159	425.3774	1.2013	24	485.8931	1440	2	2	6	5	5	20
10_District5_WSRPv2	39	156	532.5	1.2564	24	496.5769	1440	1	5	5	7	6	24
11_District0_WSRPv2	7	31	298.0645	1.129	24	498.3871	1440	2	0	3	1	1	7
11_District1_WSRPv2	28	112	529.8214	1.1964	24	494.5893	1440	2	4	1	4	10	21
11_District2_WSRPv2	9	37	387.5676	1.2432	24	527.8108	1440	0	1	3	1	0	5
11_District3_WSRPv2	13	53	332.5472	1.2264	24	485.0943	1440	4	0	2	2	2	10
11_District4_WSRPv2	38	153	477.451	1.1634	24	487.9869	1440	1	3	4	6	6	20
11_District5_WSRPv2	34	136	522.4632	1.1765	24	423.6324	1440	3	0	9	4	6	22
12_District0_WSRPv2	14	57	296.8421	1.193	24	362.1053	1440	1	3	1	3	1	9
12_District1_WSRPv2	41	164	492.8049	1.1707	24	499.0976	1440	6	1	8	9	5	29
12_District2_WSRPv2	15	61	369.3443	1.1967	24	435.1148	1440	4	0	3	1	0	8
12_District3_WSRPv2	17	71	334.4366	1.1268	24	428.0282	1440	4	1	3	1	4	13
12_District4_WSRPv2	47	190	434.7632	1.2105	24	514.0579	1440	5	5	6	4	8	28
12_District5_WSRPv2	46	187	508.0749	1.1444	24	435.8021	1440	4	8	6	7	6	31
13_District0_WSRPv2	14	58	317.069	1.1552	24	457.7586	1440	2	2	0	1	2	7
13_District1_WSRPv2	34	138	499.4565	1.1884	24	519.2246	1440	2	5	8	4	3	22
13_District2_WSRPv2	16	64	316.1719	1.1562	24	387.625	1440	2	0	4	5	1	12
13_District3_WSRPv2	19	78	345.7692	1.1538	24	452.6795	1440	4	3	2	4	5	18
13_District4_WSRPv2	40	161	435.4658	1.2547	24	519.2671	1440	10	2	11	6	9	38
13_District5_WSRPv2	42	168	464.1071	1.2619	24	457.756	1440	5	5	11	8	4	33
14_District0_WSRPv2	11	44	268.6364	1.25	24	448.6364	1440	4	2	3	1	0	10
14_District1_WSRPv2	33	134	477.0896	1.1866	24	526.6642	1440	4	4	6	8	4	26
14_District2_WSRPv2	13	55	396.8182	1.2364	24	456.4	1440	4	2	3	2	1	12
14_District3_WSRPv2	17	71	343.5211	1.1831	24	430.2394	1440	3	2	1	4	2	12
14_District4_WSRPv2	41	167	468.1437	1.1856	24	509.8563	1440	6	6	3	7	8	30
14_District5_WSRPv2	42	169	486.0355	1.2249	24	429.7515	1440	6	6	6	10	9	37
15_District0_WSRPv2	12	51	258.2353	1.1569	24	410.2941	1440	3	4	1	0	2	10
15_District1_WSRPv2	32	130	511.8462	1.1615	24	499.9462	1440	6	4	3	5	7	25
15_District2_WSRPv2	14	58	334.3966	1.1207	24	375.5	1440	4	1	4	2	0	11
15_District3_WSRPv2	19	78	320.5769	1.2436	24	388.2692	1440	4	1	6	6	1	18
15_District4_WSRPv2	40	163	413.3742	1.2025	24	519.2822	1440	4	2	2	6	7	21
15_District5_WSRPv2	39	157	495.3822	1.1783	24	477.8917	1440	3	4	3	6	5	21
16_District0_WSRPv2	13	54	287.7778	1.2407	24	425.5556	1440	3	4	2	2	0	11
16_District1_WSRPv2	32	128	506.1328	1.2422	24	448.1172	1440	6	1	6	2	2	17
16_District2_WSRPv2	13	54	424.4444	1.1667	24	441.2407	1440	2	1	3	1	3	10
16_District3_WSRPv2	17	68	344.1176	1.2059	24	387.4559	1440	4	2	2	1	4	13

Continued on next page

Table C.1 – *Continued from previous page*

Instance	# Workers	# Visits	VisitDur	Worker/Visit	WorkerHour	VistWindow	Horizon	#TimeDep					
								Sync	Overlap	Min	Max	MinMax	Total
16_District4_WSRPv2	41	166	443.5843	1.1867	24	516.6325	1440	3	4	5	9	9	30
16_District5_WSRPv2	43	175	485.1429	1.1829	24	485.8571	1440	6	5	8	2	6	27
17_District0_WSRPv2	12	48	312.5	1.1667	24	448.75	1440	3	6	2	1	0	12
17_District1_WSRPv2	31	125	532.44	1.208	24	498.048	1440	4	6	2	4	3	19
17_District2_WSRPv2	14	58	391.5517	1.1552	24	478.1897	1440	2	3	3	4	0	12
17_District3_WSRPv2	15	63	361.9048	1.2698	24	397.9683	1440	4	2	3	2	1	12
17_District4_WSRPv2	38	155	421.3548	1.2452	24	467.4645	1440	5	5	5	3	7	25
17_District5_WSRPv2	36	146	544.4178	1.1918	24	475.6301	1440	1	3	8	8	4	24
18_District0_WSRPv2	6	26	267.6923	1.3077	24	399.2308	1440	0	0	1	2	2	5
18_District1_WSRPv2	31	127	526.7717	1.2283	24	502.378	1440	6	3	5	6	7	27
18_District2_WSRPv2	8	33	331.3636	1.3333	24	377.0303	1440	2	1	0	0	1	4
18_District3_WSRPv2	12	49	396.1224	1.3265	24	475.4082	1440	2	0	1	3	4	10
18_District4_WSRPv2	36	147	435.4082	1.2517	24	514.4762	1440	5	8	9	4	5	31
18_District5_WSRPv2	31	126	544.881	1.254	24	492.7937	1440	2	6	6	4	8	26
19_District0_WSRPv2	12	49	268.1633	1.2449	24	427.3469	1440	2	0	4	2	0	8
19_District1_WSRPv2	36	146	506.7123	1.1781	24	536.4452	1440	3	7	4	7	6	27
19_District2_WSRPv2	15	62	394.3548	1.1452	24	490.5161	1440	5	2	1	2	0	10
19_District3_WSRPv2	17	69	325.8696	1.1884	24	436.4058	1440	0	4	6	3	4	17
19_District4_WSRPv2	52	210	443.0714	1.2381	24	535.2095	1440	5	7	8	11	7	38
19_District5_WSRPv2	47	191	503.089	1.1885	24	441.555	1440	7	7	10	9	10	43
1_District0_WSRPv2	18	73	303.6986	1.1781	24	427.6027	1440	6	2	4	6	0	18
1_District1_WSRPv2	44	176	490.6534	1.1932	24	539.4716	1440	6	8	10	4	9	37
1_District2_WSRPv2	19	78	390.1923	1.2051	24	500.2692	1440	4	1	5	4	2	16
1_District3_WSRPv2	22	90	341.5	1.1889	24	400	1440	2	5	2	2	3	14
1_District4_WSRPv2	51	204	431.1029	1.1961	24	495.3284	1440	6	3	10	3	9	31
1_District5_WSRPv2	49	197	489.1371	1.1929	24	469.4061	1440	3	8	11	9	6	37
20_District0_WSRPv2	13	53	290.3774	1.1509	24	392.8302	1440	2	3	3	3	1	12
20_District1_WSRPv2	33	135	511.3333	1.2222	24	509.437	1440	1	5	1	3	7	17
20_District2_WSRPv2	12	49	353.2653	1.1429	24	442.6531	1440	3	1	3	1	3	11
20_District3_WSRPv2	18	74	297.973	1.1892	24	408.8378	1440	2	3	3	1	6	15
20_District4_WSRPv2	41	165	418.7273	1.1879	24	537.3455	1440	3	3	4	5	5	20
20_District5_WSRPv2	44	178	471.7416	1.1742	24	455.264	1440	7	2	9	5	11	34
21_District0_WSRPv2	15	60	265	1.1667	24	407.75	1440	2	1	7	2	1	13
21_District1_WSRPv2	34	139	495	1.1871	24	477.446	1440	2	5	7	6	8	28
21_District2_WSRPv2	13	55	402	1.1636	24	569.4364	1440	2	1	1	1	0	5
21_District3_WSRPv2	21	84	378.2143	1.2381	24	478.75	1440	1	0	4	1	4	10
21_District4_WSRPv2	39	159	456.6038	1.2201	24	552.6415	1440	4	2	10	7	8	31
21_District5_WSRPv2	42	171	490.7018	1.2164	24	482.3977	1440	4	4	6	7	4	25
22_District0_WSRPv2	14	56	297.3214	1.25	24	454.0179	1440	1	2	2	4	6	15
22_District1_WSRPv2	33	132	494.7727	1.2576	24	449.7121	1440	4	4	4	7	4	23
22_District2_WSRPv2	16	65	359.3077	1.2462	24	417.0923	1440	1	2	1	3	1	8
22_District3_WSRPv2	18	75	327.6	1.1467	24	370.8	1440	4	1	5	3	4	17
22_District4_WSRPv2	40	162	441.9444	1.2222	24	498.2778	1440	3	4	3	11	8	29
22_District5_WSRPv2	41	165	505.4545	1.2121	24	485.4485	1440	1	3	7	5	8	24
23_District0_WSRPv2	10	42	292.8571	1.2381	24	412.8571	1440	2	0	3	0	3	8
23_District1_WSRPv2	35	141	503.7234	1.1631	24	484.3617	1440	3	6	7	4	4	24
23_District2_WSRPv2	15	62	386.8548	1.2097	24	412.3548	1440	6	0	3	3	2	14
23_District3_WSRPv2	17	69	381.3043	1.1304	24	415.4348	1440	1	3	2	2	3	11
23_District4_WSRPv2	42	168	452.4107	1.244	24	493.3929	1440	7	6	4	7	5	29
23_District5_WSRPv2	46	186	495.7258	1.2097	24	444.9946	1440	9	5	4	6	4	28
24_District0_WSRPv2	14	56	305.3571	1.1607	24	418.125	1440	3	1	1	5	2	12
24_District1_WSRPv2	31	126	535.5952	1.2222	24	466.7222	1440	8	2	6	3	5	24
24_District2_WSRPv2	10	43	392.093	1.2093	24	351.2558	1440	0	0	2	1	3	6
24_District3_WSRPv2	16	64	345	1.1562	24	480.1094	1440	2	3	4	1	3	13
24_District4_WSRPv2	41	165	429.0909	1.2061	24	512.8121	1440	5	9	6	10	4	34
24_District5_WSRPv2	36	145	542.2759	1.2345	24	461.269	1440	5	6	2	5	7	25
25_District0_WSRPv2	6	26	286.1538	1.2308	25	437.3077	1500	1	1	2	1	2	7
25_District1_WSRPv2	28	113	530.4425	1.1593	25	529.5664	1500	5	0	4	4	6	19
25_District2_WSRPv2	7	28	358.3929	1.25	25	406.6071	1500	1	1	1	1	0	4
25_District3_WSRPv2	11	46	415.4348	1.3696	25	553.3696	1500	2	1	0	2	1	6
25_District4_WSRPv2	38	154	448.1494	1.1948	25	523.2468	1500	4	3	7	9	7	30
25_District5_WSRPv2	30	123	565.6098	1.1789	25	519.935	1500	2	3	5	6	2	18
26_District0_WSRPv2	16	65	296.7692	1.2308	24	416.5385	1440	3	3	4	4	3	17
26_District1_WSRPv2	41	164	506.3415	1.2134	24	474.5061	1440	2	2	9	11	1	25
26_District2_WSRPv2	17	69	378.6957	1.2174	24	437.7101	1440	2	2	4	6	2	16
26_District3_WSRPv2	23	92	352.8261	1.2174	24	423.75	1440	5	3	7	1	1	17
26_District4_WSRPv2	52	210	462.1429	1.1905	24	513.281	1440	4	2	10	16	9	41
26_District5_WSRPv2	48	193	485.9845	1.1865	24	496.8549	1440	6	3	10	8	8	35
27_District0_WSRPv2	15	60	248.5	1.3	24	362	1440	3	3	1	3	4	14

Continued on next page

Table C.1 – *Continued from previous page*

Instance	# Workers	# Visits	VisitDur	Worker/Visit	WorkerHour	VistWindow	Horizon	#TimeDep					
								Sync	Overlap	Min	Max	MinMax	Total
27_District1_WSRPv2	34	138	512.3913	1.2174	24	539.0217	1440	1	6	4	4	12	27
27_District2_WSRPv2	14	56	356.7857	1.1429	24	499.6786	1440	1	4	0	3	2	10
27_District3_WSRPv2	15	60	299.75	1.2167	24	356.6167	1440	2	2	0	4	4	12
27_District4_WSRPv2	43	174	411.5517	1.2184	24	525.2874	1440	3	4	6	7	10	30
27_District5_WSRPv2	42	169	489.0533	1.1538	24	462.1893	1440	4	5	4	9	5	27
28_District0_WSRPv2	13	55	316.9091	1.1818	24	468.2727	1440	1	1	1	4	2	9
28_District1_WSRPv2	34	137	490.8394	1.1679	24	558.8175	1440	1	2	7	4	5	19
28_District2_WSRPv2	13	52	385.9615	1.2308	24	361.2692	1440	1	1	2	3	4	11
28_District3_WSRPv2	21	86	323.7209	1.2326	24	443.0233	1440	5	4	1	1	7	18
28_District4_WSRPv2	40	162	419.7222	1.2037	24	477.9568	1440	5	2	7	6	5	25
28_District5_WSRPv2	40	161	487.7329	1.1925	24	459.4907	1440	3	2	8	5	9	27
29_District0_WSRPv2	11	44	278.1818	1.1818	24	400.2273	1440	2	1	0	2	2	7
29_District1_WSRPv2	29	116	467.9741	1.2586	24	462.7931	1440	4	8	4	3	6	25
29_District2_WSRPv2	14	58	366.7241	1.1724	24	513.7241	1440	4	1	4	2	2	13
29_District3_WSRPv2	17	69	348.0435	1.1884	24	430.5362	1440	1	4	2	1	5	13
29_District4_WSRPv2	32	130	443.8846	1.2	24	551.7615	1440	5	5	5	0	5	20
29_District5_WSRPv2	41	166	478.4639	1.1867	24	457.3494	1440	2	11	4	5	6	28
2_District0_WSRPv2	15	60	274.5	1.2667	24	402.75	1440	4	0	1	3	3	11
2_District1_WSRPv2	40	163	507.4233	1.1902	24	456.0184	1440	6	5	9	8	3	31
2_District2_WSRPv2	16	66	348.8636	1.2273	24	426.4545	1440	1	2	1	3	6	13
2_District3_WSRPv2	21	86	350.7558	1.2093	24	452.3488	1440	5	5	3	3	0	16
2_District4_WSRPv2	47	190	454.1053	1.1842	24	503.6789	1440	5	5	8	8	5	31
2_District5_WSRPv2	50	201	492.5373	1.1891	24	458.0149	1440	4	11	6	10	11	42
30_District0_WSRPv2	9	38	285	1.1579	24	401.0526	1440	1	2	2	3	2	10
30_District1_WSRPv2	25	103	512.4757	1.233	24	439.1359	1440	5	4	5	6	3	23
30_District2_WSRPv2	11	47	392.234	1.1277	24	416.4894	1440	1	1	2	1	2	7
30_District3_WSRPv2	14	59	349.0678	1.2542	24	385.1695	1440	5	0	2	2	2	11
30_District4_WSRPv2	30	121	410.4545	1.2397	24	512.7769	1440	6	5	4	3	8	26
30_District5_WSRPv2	27	111	485	1.2162	24	484.7838	1440	6	4	5	2	8	25
3_District0_WSRPv2	13	54	296.1111	1.2222	24	429.1667	1440	3	2	1	1	2	9
3_District1_WSRPv2	33	135	532.1111	1.2074	24	546.2741	1440	4	3	3	6	9	25
3_District2_WSRPv2	16	64	389.7656	1.125	24	402.875	1440	3	1	4	1	1	10
3_District3_WSRPv2	21	85	365.1176	1.2118	24	446.1176	1440	3	2	2	5	2	14
3_District4_WSRPv2	50	200	438.075	1.19	24	517.79	1440	2	4	7	5	9	27
3_District5_WSRPv2	45	183	519.2623	1.1421	24	486.8798	1440	6	3	8	6	8	31
4_District0_WSRPv2	10	40	290.25	1.1	24	411.375	1440	1	4	0	1	1	7
4_District1_WSRPv2	33	132	560.6818	1.1667	24	510.5076	1440	4	4	3	2	4	17
4_District2_WSRPv2	10	41	377.9268	1.2683	24	457.122	1440	2	0	1	0	3	6
4_District3_WSRPv2	11	47	329.3617	1.2128	24	500.5745	1440	3	4	0	1	2	10
4_District4_WSRPv2	45	182	432.1154	1.2143	24	542.9615	1440	5	7	7	7	7	33
4_District5_WSRPv2	37	151	544.4702	1.2053	24	476.2185	1440	6	4	5	8	5	28
5_District0_WSRPv2	14	56	277.5	1.1071	24	392.1429	1440	3	3	2	1	2	11
5_District1_WSRPv2	36	144	500.4167	1.2292	24	497.0208	1440	1	2	7	5	15	30
5_District2_WSRPv2	13	53	371.8868	1.2075	24	443.9057	1440	2	5	1	2	3	13
5_District3_WSRPv2	19	76	345.1974	1.1711	24	378.5526	1440	3	5	3	2	3	16
5_District4_WSRPv2	47	188	476.4894	1.2021	24	516.7766	1440	1	7	4	6	9	27
5_District5_WSRPv2	50	201	491.5672	1.2687	24	456.5871	1440	7	3	7	7	10	34
6_District0_WSRPv2	15	60	279.5	1.2833	24	399.75	1440	1	3	4	3	0	11
6_District1_WSRPv2	38	155	463.1613	1.2	24	458.9355	1440	3	8	10	3	10	34
6_District2_WSRPv2	17	70	341.1429	1.1714	24	435.4143	1440	3	2	3	2	3	13
6_District3_WSRPv2	16	67	356.194	1.2537	24	438.806	1440	2	2	2	2	4	12
6_District4_WSRPv2	42	171	415.2632	1.2105	24	518.1053	1440	2	7	11	6	6	32
6_District5_WSRPv2	44	177	499.0678	1.1808	24	458.1299	1440	3	5	10	7	8	33
7_District0_WSRPv2	12	49	288.9796	1.2245	24	464.0816	1440	2	1	0	3	0	6
7_District1_WSRPv2	38	153	480.098	1.183	24	516.3203	1440	6	4	4	3	5	22
7_District2_WSRPv2	13	55	381.2727	1.2182	24	488.4364	1440	2	4	3	3	1	13
7_District3_WSRPv2	20	82	323.2317	1.2683	24	395.122	1440	6	2	7	1	1	17
7_District4_WSRPv2	41	167	443.0838	1.1856	24	484.7485	1440	7	3	5	4	10	29
7_District5_WSRPv2	45	181	469.9724	1.1713	24	471.663	1440	4	6	5	9	13	37
8_District0_WSRPv2	12	49	274.2857	1.2857	24	425.5102	1440	3	3	3	1	3	13
8_District1_WSRPv2	32	130	533.6538	1.1769	24	485.1769	1440	1	4	7	6	4	22
8_District2_WSRPv2	13	53	341.8868	1.1509	24	371.8868	1440	1	0	4	3	1	9
8_District3_WSRPv2	18	74	348.2432	1.2027	24	485.2703	1440	5	1	4	4	5	19
8_District4_WSRPv2	41	166	428.2229	1.1867	24	456.2711	1440	7	7	6	3	5	28
8_District5_WSRPv2	40	162	508.3333	1.216	24	491.9383	1440	4	7	9	4	7	31
9_District0_WSRPv2	12	51	301.1765	1.3137	24	399.4118	1440	0	2	1	2	3	8
9_District1_WSRPv2	31	124	519.0726	1.2339	24	471.9435	1440	3	3	5	3	5	19
9_District2_WSRPv2	13	53	428.7736	1.2075	24	370.6038	1440	1	4	2	2	1	10
9_District3_WSRPv2	22	89	359.8315	1.2022	24	389.2247	1440	2	5	5	2	3	17

Continued on next page

Table C.1 – *Continued from previous page*

Instance	# Workers	# Visits	VisitDur	Worker/Visit	WorkerHour	VistWindow	Horizon	#TimeDep					
								Sync	Overlap	Min	Max	MinMax	Total
9_District4_WSRPv2	44	178	423.6236	1.1966	24	516.7247	1440	6	3	9	5	8	31
9_District5_WSRPv2	37	151	522.4172	1.1854	24	470.3642	1440	1	5	4	9	6	25
Solomon's Instances (Sol)													
C101_100t_20w_WSRP	20	100	90	1.04	20.6	60.76	1236	1	0	1	3	0	5
C101_25t_5w_WSRP	5	25	90	1.2	20.6	60.44	1236	0	0	0	1	0	1
C101_50t_10w_WSRP	10	50	90	1.18	20.6	60.14	1236	0	0	1	2	0	3
C102_100t_20w_WSRP	20	100	90	1.04	20.6	325.69	1236	7	4	1	3	0	15
C102_25t_5w_WSRP	5	25	90	1.2	20.6	359.44	1236	0	1	0	1	0	2
C102_50t_10w_WSRP	10	50	90	1.18	20.6	336.6	1236	0	5	1	2	0	8
C103_100t_20w_WSRP	20	100	90	1.04	20.6	588.49	1236	9	5	1	3	0	18
C103_25t_5w_WSRP	5	25	90	1.2	20.6	611.6	1236	0	2	0	1	0	3
C103_50t_10w_WSRP	10	50	90	1.18	20.6	590.44	1236	1	5	1	2	0	9
C104_100t_20w_WSRP	20	100	90	1.04	20.6	852.94	1236	11	5	1	3	0	20
C104_25t_5w_WSRP	5	25	90	1.2	20.6	781.48	1236	0	3	0	1	0	4
C104_50t_10w_WSRP	10	50	90	1.18	20.6	908.04	1236	1	5	1	2	0	9
C105_100t_20w_WSRP	20	100	90	1.04	20.6	121.61	1236	1	3	1	3	0	8
C105_25t_5w_WSRP	5	25	90	1.2	20.6	121.08	1236	0	2	0	1	0	3
C105_50t_10w_WSRP	10	50	90	1.18	20.6	120.38	1236	1	1	1	2	0	5
C106_100t_20w_WSRP	20	100	90	1.04	20.6	156.15	1236	1	1	1	3	0	6
C106_25t_5w_WSRP	5	25	90	1.2	20.6	73.72	1236	0	0	0	1	0	1
C106_50t_10w_WSRP	10	50	90	1.18	20.6	94.36	1236	0	0	1	2	0	3
C107_100t_20w_WSRP	20	100	90	1.04	20.6	180	1236	2	3	1	3	0	9
C107_25t_5w_WSRP	5	25	90	1.2	20.6	180	1236	0	2	0	1	0	3
C107_50t_10w_WSRP	10	50	90	1.18	20.6	180	1236	1	1	1	2	0	5
C108_100t_20w_WSRP	20	100	90	1.04	20.6	243.28	1236	4	4	1	3	0	12
C108_25t_5w_WSRP	5	25	90	1.2	20.6	242.16	1236	0	2	0	1	0	3
C108_50t_10w_WSRP	10	50	90	1.18	20.6	240.78	1236	1	1	1	2	0	5
C109_100t_20w_WSRP	20	100	90	1.04	20.6	360	1236	6	4	1	3	0	14
C109_25t_5w_WSRP	5	25	90	1.2	20.6	360	1236	0	2	0	1	0	3
C109_50t_10w_WSRP	10	50	90	1.18	20.6	360	1236	1	2	1	2	0	6
C201_100t_20w_WSRP	20	100	90	1.04	56.5	160	3390	0	1	1	3	0	5
C201_25t_5w_WSRP	5	25	90	1.2	56.5	160	3390	0	1	0	1	0	2
C201_50t_10w_WSRP	10	50	90	1.18	56.5	160	3390	0	2	1	2	0	5
C202_100t_20w_WSRP	20	100	90	1.04	56.5	937.74	3390	6	4	1	3	0	14
C202_25t_5w_WSRP	5	25	90	1.2	56.5	1032.28	3390	0	2	0	1	0	3
C202_50t_10w_WSRP	10	50	90	1.18	56.5	969.42	3390	0	5	1	2	0	8
C203_100t_20w_WSRP	20	100	90	1.04	56.5	1714.82	3390	9	5	1	3	0	18
C203_25t_5w_WSRP	5	25	90	1.2	56.5	1778.6	3390	0	3	0	1	0	4
C203_50t_10w_WSRP	10	50	90	1.18	56.5	1716.36	3390	1	5	1	2	0	9
C204_100t_20w_WSRP	20	100	90	1.04	56.5	2492.58	3390	11	5	1	3	0	20
C204_25t_5w_WSRP	5	25	90	1.2	56.5	2277.4	3390	0	3	0	1	0	4
C204_50t_10w_WSRP	10	50	90	1.18	56.5	2650.24	3390	1	5	1	2	0	9
C205_100t_20w_WSRP	20	100	90	1.04	56.5	320	3390	5	3	1	3	0	12
C205_25t_5w_WSRP	5	25	90	1.2	56.5	320	3390	0	1	0	1	0	2
C205_50t_10w_WSRP	10	50	90	1.18	56.5	320	3390	0	2	1	2	0	5
C206_100t_20w_WSRP	20	100	90	1.04	56.5	486.64	3390	7	3	1	3	0	14
C206_25t_5w_WSRP	5	25	90	1.2	56.5	464.52	3390	0	2	0	1	0	3
C206_50t_10w_WSRP	10	50	90	1.18	56.5	480.48	3390	0	5	1	2	0	8
C207_100t_20w_WSRP	20	100	90	1.04	56.5	612.32	3390	6	3	1	3	0	13
C207_25t_5w_WSRP	5	25	90	1.2	56.5	742	3390	0	2	0	1	0	3
C207_50t_10w_WSRP	10	50	90	1.18	56.5	790.8	3390	1	4	1	2	0	8
C208_100t_20w_WSRP	20	100	90	1.04	56.5	640	3390	9	3	1	3	0	16
C208_25t_5w_WSRP	5	25	90	1.2	56.5	640	3390	0	2	0	1	0	3
C208_50t_10w_WSRP	10	50	90	1.18	56.5	640	3390	1	5	1	2	0	9
R101_100t_20w_WSRP	20	100	10	1.04	3.8333	10	230	1	0	1	3	0	5
R101_25t_5w_WSRP	5	25	10	1.2	3.8333	10	230	0	0	0	1	0	1
R101_50t_10w_WSRP	10	50	10	1.18	3.8333	10	230	0	0	1	2	0	3
R102_100t_20w_WSRP	20	100	10	1.04	3.8333	57.39	230	6	4	1	3	0	14
R102_25t_5w_WSRP	5	25	10	1.2	3.8333	63.44	230	0	1	0	1	0	2
R102_50t_10w_WSRP	10	50	10	1.18	3.8333	59.24	230	0	5	1	2	0	8
R103_100t_20w_WSRP	20	100	10	1.04	3.8333	102.99	230	9	5	1	3	0	18
R103_25t_5w_WSRP	5	25	10	1.2	3.8333	106.88	230	0	2	0	1	0	3
R103_50t_10w_WSRP	10	50	10	1.18	3.8333	102.62	230	1	5	1	2	0	9
R104_100t_20w_WSRP	20	100	10	1.04	3.8333	148.31	230	11	5	1	3	0	20
R104_25t_5w_WSRP	5	25	10	1.2	3.8333	136.64	230	0	3	0	1	0	4
R104_50t_10w_WSRP	10	50	10	1.18	3.8333	157.4	230	1	5	1	2	0	9
R105_100t_20w_WSRP	20	100	10	1.04	3.8333	30	230	6	0	1	3	0	10
R105_25t_5w_WSRP	5	25	10	1.2	3.8333	30	230	0	0	0	1	0	1

Continued on next page

Table C.1 – *Continued from previous page*

Instance	# Workers	# Visits	VisitDur	Worker/Visit	WorkerHour	VistWindow	Horizon	#TimeDep					
								Sync	Overlap	Min	Max	MinMax	Total
R105_50t_10w_WSRP	10	50	10	1.18	3.8333	30	230	0	1	1	2	0	4
R106_100t_20w_WSRP	20	100	10	1.04	3.8333	72.39	230	9	4	1	3	0	17
R106_25t_5w_WSRP	5	25	10	1.2	3.8333	77.84	230	0	1	0	1	0	2
R106_50t_10w_WSRP	10	50	10	1.18	3.8333	74.04	230	0	5	1	2	0	8
R107_100t_20w_WSRP	20	100	10	1.04	3.8333	112.99	230	10	5	1	3	0	19
R107_25t_5w_WSRP	5	25	10	1.2	3.8333	116.48	230	0	2	0	1	0	3
R107_50t_10w_WSRP	10	50	10	1.18	3.8333	112.62	230	1	5	1	2	0	9
R108_100t_20w_WSRP	20	100	10	1.04	3.8333	153.31	230	11	5	1	3	0	20
R108_25t_5w_WSRP	5	25	10	1.2	3.8333	143.04	230	0	3	0	1	0	4
R108_50t_10w_WSRP	10	50	10	1.18	3.8333	161.4	230	1	5	1	2	0	9
R109_100t_20w_WSRP	20	100	10	1.04	3.8333	58.89	230	6	2	1	3	0	12
R109_25t_5w_WSRP	5	25	10	1.2	3.8333	58.36	230	0	1	0	1	0	2
R109_50t_10w_WSRP	10	50	10	1.18	3.8333	58.94	230	0	3	1	2	0	6
R110_100t_20w_WSRP	20	100	10	1.04	3.8333	86.5	230	9	4	1	3	0	17
R110_25t_5w_WSRP	5	25	10	1.2	3.8333	83.28	230	0	3	0	1	0	4
R110_50t_10w_WSRP	10	50	10	1.18	3.8333	86.44	230	1	3	1	2	0	7
R111_100t_20w_WSRP	20	100	10	1.04	3.8333	93.1	230	10	5	1	3	0	19
R111_25t_5w_WSRP	5	25	10	1.2	3.8333	93.72	230	0	2	0	1	0	3
R111_50t_10w_WSRP	10	50	10	1.18	3.8333	95.46	230	1	5	1	2	0	9
R112_100t_20w_WSRP	20	100	10	1.04	3.8333	117.64	230	12	5	1	3	0	21
R112_25t_5w_WSRP	5	25	10	1.2	3.8333	116.44	230	0	3	0	1	0	4
R112_50t_10w_WSRP	10	50	10	1.18	3.8333	117.76	230	1	5	1	2	0	9
R201_100t_20w_WSRP	20	100	10	1.04	16.6667	115.96	1000	2	0	1	3	0	6
R201_25t_5w_WSRP	5	25	10	1.2	16.6667	113.72	1000	0	0	0	1	0	1
R201_50t_10w_WSRP	10	50	10	1.18	16.6667	116.46	1000	0	0	1	2	0	3
R202_100t_20w_WSRP	20	100	10	1.04	16.6667	328.81	1000	7	4	1	3	0	15
R202_25t_5w_WSRP	5	25	10	1.2	16.6667	352.56	1000	0	1	0	1	0	2
R202_50t_10w_WSRP	10	50	10	1.18	16.6667	339.96	1000	0	5	1	2	0	8
R203_100t_20w_WSRP	20	100	10	1.04	16.6667	541.66	1000	9	5	1	3	0	18
R203_25t_5w_WSRP	5	25	10	1.2	16.6667	554.96	1000	0	2	0	1	0	3
R203_50t_10w_WSRP	10	50	10	1.18	16.6667	541.54	1000	1	5	1	2	0	9
R204_100t_20w_WSRP	20	100	10	1.04	16.6667	751.26	1000	11	5	1	3	0	20
R204_25t_5w_WSRP	5	25	10	1.2	16.6667	694.48	1000	0	3	0	1	0	4
R204_50t_10w_WSRP	10	50	10	1.18	16.6667	794.32	1000	1	5	1	2	0	9
R205_100t_20w_WSRP	20	100	10	1.04	16.6667	240	1000	6	2	1	3	0	12
R205_25t_5w_WSRP	5	25	10	1.2	16.6667	240	1000	0	1	0	1	0	2
R205_50t_10w_WSRP	10	50	10	1.18	16.6667	240	1000	0	2	1	2	0	5
R206_100t_20w_WSRP	20	100	10	1.04	16.6667	422.39	1000	9	4	1	3	0	17
R206_25t_5w_WSRP	5	25	10	1.2	16.6667	444.64	1000	0	1	0	1	0	2
R206_50t_10w_WSRP	10	50	10	1.18	16.6667	429.64	1000	0	5	1	2	0	8
R207_100t_20w_WSRP	20	100	10	1.04	16.6667	602.99	1000	10	5	1	3	0	19
R207_25t_5w_WSRP	5	25	10	1.2	16.6667	617.68	1000	0	2	0	1	0	3
R207_50t_10w_WSRP	10	50	10	1.18	16.6667	602.62	1000	1	5	1	2	0	9
R208_100t_20w_WSRP	20	100	10	1.04	16.6667	783.31	1000	11	5	1	3	0	20
R208_25t_5w_WSRP	5	25	10	1.2	16.6667	733.84	1000	0	3	0	1	0	4
R208_50t_10w_WSRP	10	50	10	1.18	16.6667	819.4	1000	1	5	1	2	0	9
R209_100t_20w_WSRP	20	100	10	1.04	16.6667	349.5	1000	8	2	1	3	0	14
R209_25t_5w_WSRP	5	25	10	1.2	16.6667	332.72	1000	0	3	0	1	0	4
R209_50t_10w_WSRP	10	50	10	1.18	16.6667	351.08	1000	1	3	1	2	0	7
R210_100t_20w_WSRP	20	100	10	1.04	16.6667	383.27	1000	9	4	1	3	0	17
R210_25t_5w_WSRP	5	25	10	1.2	16.6667	385.88	1000	0	1	0	1	0	2
R210_50t_10w_WSRP	10	50	10	1.18	16.6667	390.06	1000	0	5	1	2	0	8
R211_100t_20w_WSRP	20	100	10	1.04	16.6667	471.94	1000	12	5	1	3	0	21
R211_25t_5w_WSRP	5	25	10	1.2	16.6667	467.48	1000	0	3	0	1	0	4
R211_50t_10w_WSRP	10	50	10	1.18	16.6667	472.92	1000	1	5	1	2	0	9
RC101_100t_20w_WSRP	20	100	10	1.04	4	30	240	4	1	1	3	0	9
RC101_25t_5w_WSRP	5	25	10	1.2	4	30	240	0	1	0	1	0	2
RC101_50t_10w_WSRP	10	50	10	1.18	4	30	240	0	1	1	2	0	4
RC102_100t_20w_WSRP	20	100	10	1.04	4	71.46	240	8	4	1	3	0	16
RC102_25t_5w_WSRP	5	25	10	1.2	4	75.4	240	0	1	0	1	0	2
RC102_50t_10w_WSRP	10	50	10	1.18	4	71.08	240	0	5	1	2	0	8
RC103_100t_20w_WSRP	20	100	10	1.04	4	112.5	240	10	5	1	3	0	19
RC103_25t_5w_WSRP	5	25	10	1.2	4	113.92	240	0	2	0	1	0	3
RC103_50t_10w_WSRP	10	50	10	1.18	4	108.8	240	1	5	1	2	0	9
RC104_100t_20w_WSRP	20	100	10	1.04	4	154.6	240	11	5	1	3	0	20
RC104_25t_5w_WSRP	5	25	10	1.2	4	140.24	240	0	3	0	1	0	4
RC104_50t_10w_WSRP	10	50	10	1.18	4	156.54	240	1	5	1	2	0	9
RC105_100t_20w_WSRP	20	100	10	1.04	4	54.33	240	7	3	1	3	0	14
RC105_25t_5w_WSRP	5	25	10	1.2	4	55.28	240	0	1	0	1	0	2

Continued on next page

Table C.1 – *Continued from previous page*

Instance	# Workers	# Visits	VisitDur	Worker/Visit	WorkerHour	VistWindow	Horizon	#TimeDep					
								Sync	Overlap	Min	Max	MinMax	Total
RC105_50t_10w_WSRP	10	50	10	1.18	4	56.38	240	0	5	1	2	0	8
RC106_100t_20w_WSRP	20	100	10	1.04	4	60	240	6	2	1	3	0	12
RC106_25t_5w_WSRP	5	25	10	1.2	4	60	240	0	1	0	1	0	2
RC106_50t_10w_WSRP	10	50	10	1.18	4	60	240	1	3	1	2	0	7
RC107_100t_20w_WSRP	20	100	10	1.04	4	88.21	240	9	4	1	3	0	17
RC107_25t_5w_WSRP	5	25	10	1.2	4	85.96	240	0	3	0	1	0	4
RC107_50t_10w_WSRP	10	50	10	1.18	4	88.1	240	1	3	1	2	0	7
RC108_100t_20w_WSRP	20	100	10	1.04	4	112.33	240	12	5	1	3	0	21
RC108_25t_5w_WSRP	5	25	10	1.2	4	110.48	240	0	3	0	1	0	4
RC108_50t_10w_WSRP	10	50	10	1.18	4	111.62	240	1	5	1	2	0	9
RC201_100t_20w_WSRP	20	100	10	1.04	16	120	960	2	0	1	3	0	6
RC201_25t_5w_WSRP	5	25	10	1.2	16	120	960	0	0	0	1	0	1
RC201_50t_10w_WSRP	10	50	10	1.18	16	120	960	0	0	1	2	0	3
RC202_100t_20w_WSRP	20	100	10	1.04	16	318.96	960	6	4	1	3	0	14
RC202_25t_5w_WSRP	5	25	10	1.2	16	341.8	960	0	1	0	1	0	2
RC202_50t_10w_WSRP	10	50	10	1.18	16	324.88	960	0	5	1	2	0	8
RC203_100t_20w_WSRP	20	100	10	1.04	16	517.5	960	9	5	1	3	0	18
RC203_25t_5w_WSRP	5	25	10	1.2	16	531.52	960	0	2	0	1	0	3
RC203_50t_10w_WSRP	10	50	10	1.18	16	513.8	960	1	5	1	2	0	9
RC204_100t_20w_WSRP	20	100	10	1.04	16	717.1	960	11	5	1	3	0	20
RC204_25t_5w_WSRP	5	25	10	1.2	16	658.64	960	0	3	0	1	0	4
RC204_50t_10w_WSRP	10	50	10	1.18	16	750.54	960	1	5	1	2	0	9
RC205_100t_20w_WSRP	20	100	10	1.04	16	223.06	960	5	3	1	3	0	12
RC205_25t_5w_WSRP	5	25	10	1.2	16	227.76	960	0	1	0	1	0	2
RC205_50t_10w_WSRP	10	50	10	1.18	16	230.5	960	0	4	1	2	0	7
RC206_100t_20w_WSRP	20	100	10	1.04	16	240	960	6	2	1	3	0	12
RC206_25t_5w_WSRP	5	25	10	1.2	16	240	960	0	1	0	1	0	2
RC206_50t_10w_WSRP	10	50	10	1.18	16	240	960	0	2	1	2	0	5
RC207_100t_20w_WSRP	20	100	10	1.04	16	349.5	960	9	3	1	3	0	16
RC207_25t_5w_WSRP	5	25	10	1.2	16	332.72	960	0	3	0	1	0	4
RC207_50t_10w_WSRP	10	50	10	1.18	16	351.08	960	1	3	1	2	0	7
RC208_100t_20w_WSRP	20	100	10	1.04	16	471.93	960	12	5	1	3	0	21
RC208_25t_5w_WSRP	5	25	10	1.2	16	467.44	960	0	3	0	1	0	4
RC208_50t_10w_WSRP	10	50	10	1.18	16	472.9	960	1	5	1	2	0	9
Home Healthcare Instances (HHC)													
hh_00_P0	15	153	31.9608	1	23	106.4052	1380	2	1	0	0	0	3
ll1_00_P0	9	106	24.6226	1	23	65.6698	1380	0	0	0	0	0	0
ll1_01_P0	9	106	24.6226	1	23	65.6698	1380	1	0	0	0	0	1
ll1_02_P0	9	106	24.6226	1	23	65.6698	1380	0	0	0	0	0	0
ll1_03_P0	9	106	24.6226	1	23	65.6698	1380	0	0	0	0	0	0
ll1_04_P0	9	106	24.6226	1	23	65.6698	1380	0	0	0	0	0	0
ll1_05_P0	9	106	24.6226	1	23	65.6698	1380	0	1	0	0	0	1
ll1_06_P0	9	106	24.6226	1	23	65.6698	1380	7	0	0	0	0	7
ll1_07_P0	9	106	24.6226	1	23	65.6698	1380	0	0	0	0	0	0
ll2_00_P0	7	60	30.0333	1	23	58.3333	1380	0	0	0	0	0	0
ll3_00_P0	7	60	30.05	1	23	58.2333	1380	0	0	0	0	0	0
Vehicle Routing Problem with Time Window Instances (Mov)													
test150-0-0-0-0_d0_tw0WSRP	103	150	20.4667	1.08	8	480	480	11	15	4	4	3	37
test150-0-0-0-0_d0_tw1WSRP	103	150	20.4667	1.08	8	160	480	3	3	4	4	3	17
test150-0-0-0-0_d0_tw2WSRP	103	150	20.4667	1.08	8	127.2667	480	3	3	4	4	3	17
test150-0-0-0-0_d0_tw3WSRP	103	150	20.4667	1.08	8	100	480	3	3	4	4	3	17
test150-0-0-0-0_d0_tw4WSRP	103	150	20.4667	1.08	8	155.9333	480	5	9	4	4	3	25
test250-0-0-0-0_d0_tw0WSRP	171	250	20.44	1.112	8	480	480	18	17	7	4	7	53
test250-0-0-0-0_d0_tw1WSRP	171	250	20.44	1.112	8	160	480	6	5	7	4	7	29
test250-0-0-0-0_d0_tw2WSRP	171	250	20.44	1.112	8	127.12	480	6	5	7	4	7	29
test250-0-0-0-0_d0_tw3WSRP	171	250	20.44	1.112	8	100	480	6	5	7	4	7	29
test250-0-0-0-0_d0_tw4WSRP	171	250	20.44	1.112	8	154.4	480	4	6	7	4	7	28
test50-0-0-0-0_d0_tw0WSRP	38	50	22.6	1.08	8	480	480	5	5	1	2	0	13
test50-0-0-0-0_d0_tw1WSRP	38	50	22.6	1.08	8	160	480	3	1	1	2	0	7
test50-0-0-0-0_d0_tw2WSRP	38	50	22.6	1.08	8	128.4	480	3	1	1	2	0	7
test50-0-0-0-0_d0_tw3WSRP	38	50	22.6	1.08	8	100	480	3	1	1	2	0	7
test50-0-0-0-0_d0_tw4WSRP	38	50	22.6	1.08	8	152	480	2	4	1	2	0	9

Appendix D

Results RDCR to Solve WSRP with Time-dependent Activities Constraints

Table D.1: Results RDCR to solve WSRP with time-dependent activities constraints.

Instance	Set	Solver	GHI	RDCR	GHI ratio	RDCR ratio
1_District0	Sec	514,601,438,498	518,262,674,109	447,709,682,909	20%	4%
1_District1	Sec	81,547,125,693,056	57,515,558,599,736	62,938,903,604,979	0%	10%
1_District2	Sec	980,219,980,998	833,211,902,031	800,903,345,888	24%	19%
1_District3	Sec	2,555,082,025,908	1,505,646,454,671	1,370,676,003,209	46%	33%
1_District4	Sec	-	78,076,709,551,623	88,190,472,017,848	0%	13%
1_District5	Sec	-	98,873,046,645,006	102,358,835,843,071	5%	9%
10_District0	Sec	71,563,566,909	106,514,181,491	78,088,235,511	49%	9%
10_District1	Sec	15,339,521,436,005	9,001,482,305,058	8,953,332,169,764	8%	7%
10_District2	Sec	27,771,011,394	58,625,462,521	62,649,955,234	111%	126%
10_District3	Sec	174,846,839,902	209,751,898,949	168,523,200,303	51%	21%
10_District4	Sec	58,077,637,594,031	23,282,973,667,202	24,848,955,610,201	9%	17%
10_District5	Sec	73,090,997,319,600	40,589,090,305,550	41,350,955,424,765	8%	10%
11_District0	Sec	3,046,672,128	5,544,047,419	4,528,490,787	82%	49%
11_District1	Sec	12,181,331,635,200	7,039,742,788,663	7,096,389,603,480	14%	15%
11_District2	Sec	9,214,556,192	16,733,934,536	15,701,676,953	82%	70%
11_District3	Sec	61,560,228,131	94,943,748,043	80,429,173,240	56%	32%
11_District4	Sec	50,939,600,391,450	25,566,635,276,064	25,349,028,368,524	12%	11%
11_District5	Sec	31,007,719,540,800	17,249,049,875,800	18,767,359,667,252	9%	18%
12_District0	Sec	115,036,288,620	153,991,002,969	127,119,133,946	34%	11%
12_District1	Sec	-	39,368,813,493,490	42,847,307,655,863	4%	13%
12_District2	Sec	164,768,096,901	235,123,516,633	197,832,919,167	54%	29%
12_District3	Sec	430,939,822,091	386,856,589,843	291,304,926,933	55%	17%
12_District4	Sec	-	59,129,219,661,290	70,125,934,529,756	0%	19%
12_District5	Sec	-	76,568,915,156,315	76,967,348,254,475	14%	14%
13_District0	Sec	154,315,121,626	200,168,900,259	163,180,352,402	30%	6%
13_District1	Sec	32,987,015,561,700	16,442,120,749,780	17,008,532,953,149	1%	4%
13_District2	Sec	148,613,412,428	193,620,112,937	148,951,554,097	45%	12%
13_District3	Sec	802,948,902,933	698,181,667,824	562,624,058,279	62%	31%
13_District4	Sec	-	34,000,815,348,145	35,034,996,875,949	13%	17%
13_District5	Sec	92,802,510,259,200	50,699,859,698,755	52,060,024,981,214	18%	22%
14_District0	Sec	34,977,146,773	45,850,927,116	33,613,834,195	36%	0%
14_District1	Sec	26,647,661,279,250	14,254,826,758,086	12,552,617,606,951	15%	1%
14_District2	Sec	90,165,619,841	131,633,438,179	118,697,151,642	46%	32%
14_District3	Sec	375,574,682,503	403,729,007,063	396,715,473,750	47%	45%
14_District4	Sec	78,479,954,061,290	35,559,983,065,796	40,489,428,035,134	6%	21%
14_District5	Sec	-	53,900,420,556,896	54,735,685,475,091	21%	23%
15_District0	Sec	42,188,641,727	66,146,782,173	50,324,341,525	57%	19%
15_District1	Sec	23,707,857,420,000	12,711,526,556,580	14,096,121,539,628	3%	14%
15_District2	Sec	67,421,894,827	119,992,921,511	126,283,911,227	78%	87%

Continued on next page

Table D.1 – *Continued from previous page*

Instance	Set	Solver	GHI	RDCR	GHI ratio	RDCR ratio
15_District3	Sec	919,776,368,306	612,534,148,647	587,319,527,685	32%	27%
15_District4	Sec	-	26,167,980,969,582	29,088,282,075,298	11%	23%
15_District5	Sec	61,259,711,704,425	32,316,119,579,039	33,644,916,923,816	13%	17%
16_District0	Sec	179,548,714,933	134,797,464,599	123,630,563,139	12%	2%
16_District1	Sec	25,024,848,793,229	13,544,030,449,928	13,634,850,775,453	7%	7%
16_District2	Sec	94,504,646,914	136,369,984,551	115,642,403,376	44%	22%
16_District3	Sec	446,106,615,226	310,893,633,571	274,516,046,341	48%	30%
16_District4	Sec	71,942,492,994,075	32,015,923,640,901	34,291,648,453,207	10%	17%
16_District5	Sec	67,132,893,394,884	49,863,031,314,373	54,172,469,228,640	3%	12%
17_District0	Sec	60,633,779,564	80,339,019,702	64,716,316,035	32%	7%
17_District1	Sec	22,293,898,242,188	12,781,834,998,535	12,347,609,308,417	6%	2%
17_District2	Sec	111,787,046,907	156,181,089,017	142,805,377,619	40%	28%
17_District3	Sec	266,435,015,651	286,459,175,917	256,991,305,382	60%	44%
17_District4	Sec	51,866,075,621,566	20,922,289,478,740	23,054,910,631,705	7%	18%
17_District5	Sec	34,911,929,918,411	26,532,859,320,679	27,685,527,855,707	8%	13%
18_District0	Sec	2,341,527,489	3,521,808,557	2,519,997,332	50%	8%
18_District1	Sec	18,446,554,645,865	14,328,913,847,618	14,806,048,324,150	3%	6%
18_District2	Sec	4,417,358,464	8,630,147,572	8,917,818,469	95%	102%
18_District3	Sec	73,407,720,408	93,021,012,559	83,498,616,134	31%	18%
18_District4	Sec	-	19,659,368,358,475	21,221,048,343,547	2%	10%
18_District5	Sec	25,068,488,775,750	13,672,716,328,158	14,976,985,140,375	6%	17%
19_District0	Sec	70,282,556,413	64,434,270,618	52,251,567,963	23%	0%
19_District1	Sec	43,257,877,194,600	23,652,212,368,612	23,180,723,962,845	4%	2%
19_District2	Sec	153,280,470,263	219,680,866,791	187,711,686,914	43%	22%
19_District3	Sec	427,355,363,992	346,757,448,839	322,524,813,689	42%	32%
19_District4	Sec	-	113,859,513,555,083	128,960,925,276,457	5%	19%
19_District5	Sec	162,778,290,538,750	95,534,597,869,555	92,494,639,683,456	16%	12%
2_District0	Sec	240,521,004,257	203,347,903,445	177,891,077,243	38%	21%
2_District1	Sec	76,298,011,317,000	40,571,278,712,927	41,756,330,687,371	0%	3%
2_District2	Sec	395,580,450,451	281,947,676,353	303,739,646,739	29%	39%
2_District3	Sec	2,263,843,304,100	1,178,651,432,556	920,821,431,463	33%	4%
2_District4	Sec	141,036,256,870,603	68,080,523,243,923	67,589,580,076,667	16%	15%
2_District5	Sec	207,484,128,411,053	112,807,731,538,713	118,657,446,131,747	18%	24%
20_District0	Sec	98,421,186,256	110,769,617,877	92,857,415,499	19%	0%
20_District1	Sec	24,723,996,266,847	18,197,730,346,113	18,315,034,396,858	6%	6%
20_District2	Sec	27,378,465,012	54,688,947,926	50,117,372,729	100%	83%
20_District3	Sec	498,973,322,172	470,291,321,930	369,228,829,359	43%	12%
20_District4	Sec	67,841,287,816,567	28,044,909,810,918	31,060,078,155,475	9%	20%
20_District5	Sec	71,821,062,163,723	58,730,152,299,959	59,047,521,325,582	20%	21%
21_District0	Sec	144,801,663,745	147,347,736,598	131,272,530,147	22%	9%
21_District1	Sec	33,249,102,675,975	17,166,915,036,774	18,955,481,913,335	2%	12%
21_District2	Sec	62,666,013,838	120,541,044,420	108,413,021,592	92%	73%
21_District3	Sec	1,445,196,400,668	1,297,422,147,669	956,161,573,105	45%	7%
21_District4	Sec	-	27,914,982,366,636	31,944,962,210,637	8%	23%
21_District5	Sec	98,695,460,825,325	53,568,470,949,871	56,146,030,999,419	17%	22%
22_District0	Sec	138,906,549,663	165,773,666,324	151,095,789,309	19%	9%
22_District1	Sec	29,248,434,309,600	15,243,631,318,257	17,349,191,811,548	8%	23%
22_District2	Sec	357,571,787,137	324,661,427,161	260,369,198,348	43%	15%
22_District3	Sec	490,512,412,789	474,713,951,444	328,727,467,403	66%	15%
22_District4	Sec	-	30,893,669,654,101	33,985,536,310,874	3%	14%
22_District5	Sec	84,315,534,373,125	44,875,375,329,000	48,151,990,030,799	0%	8%
23_District0	Sec	32,906,581,132	40,743,361,288	34,209,421,078	24%	4%
23_District1	Sec	35,053,516,275,858	17,469,640,737,231	20,099,409,112,987	0%	15%
23_District2	Sec	183,871,311,139	257,681,688,763	260,422,982,480	40%	42%
23_District3	Sec	329,850,892,524	365,950,771,984	287,808,645,430	46%	15%
23_District4	Sec	-	38,298,731,215,190	38,504,039,316,941	11%	11%
23_District5	Sec	-	77,953,647,815,762	85,745,830,926,258	10%	21%
24_District0	Sec	105,830,236,823	151,863,618,724	124,181,531,822	43%	17%
24_District1	Sec	-	12,692,918,197,764	13,338,761,613,824	8%	13%
24_District2	Sec	26,055,867,879	40,494,552,634	34,308,437,456	55%	32%
24_District3	Sec	306,518,541,155	241,943,658,267	184,659,485,366	52%	16%
24_District4	Sec	69,365,291,118,750	26,737,166,769,864	29,409,864,299,255	15180%	16708%
24_District5	Sec	-	28,479,935,686,410	31,728,250,870,473	15%	28%
25_District0	Sec	1,255,504,790	2,795,528,713	1,813,255,336	123%	44%
25_District1	Sec	12,146,586,587,100	6,324,900,485,757	6,689,231,492,002	1%	6%
25_District2	Sec	2,430,507,031	4,472,855,514	3,377,315,217	84%	39%
25_District3	Sec	54,825,213,021	70,786,983,265	69,030,320,720	29%	26%
25_District4	Sec	51,348,511,229,400	22,271,373,878,427	23,017,626,202,043	7%	11%
25_District5	Sec	20,038,848,410,250	11,858,978,750,864	11,279,275,134,050	8%	2%
26_District0	Sec	363,306,988,154	291,402,864,347	223,285,101,758	31%	0%
26_District1	Sec	82,246,568,310,323	43,337,058,654,901	47,062,066,270,846	3%	12%
26_District2	Sec	282,951,685,653	432,405,593,957	401,061,156,386	53%	42%
26_District3	Sec	-	1,711,555,856,290	1,440,017,664,619	46%	23%

Continued on next page

Table D.1 – *Continued from previous page*

Instance	Set	Solver	GHI	RDCR	GHI ratio	RDCR ratio
26_District4	Sec	-	116,616,244,902,672	126,312,939,621,475	7%	16%
26_District5	Sec	-	88,864,369,188,667	90,236,791,873,733	16%	18%
27_District0	Sec	162,479,691,641	196,822,644,722	162,191,094,546	21%	0%
27_District1	Sec	-	19,532,184,104,779	20,449,351,862,605	6%	11%
27_District2	Sec	107,984,714,517	97,204,084,370	89,940,537,761	39%	29%
27_District3	Sec	147,102,221,428	188,834,841,381	141,505,776,872	62%	21%
27_District4	Sec	89,970,323,684,850	34,983,506,435,735	40,877,321,398,123	4%	21%
27_District5	Sec	55,449,486,241,964	43,125,961,055,733	43,364,366,959,407	9%	10%
28_District0	Sec	102,108,349,094	139,949,789,802	116,920,632,288	37%	15%
28_District1	Sec	30,159,721,225,350	15,350,085,980,159	15,896,428,627,470	0%	4%
28_District2	Sec	64,692,968,024	104,194,611,486	109,196,777,028	61%	69%
28_District3	Sec	1,553,657,012,688	1,076,992,212,853	988,089,332,927	34%	23%
28_District4	Sec	62,657,425,341,000	25,831,106,950,400	29,828,569,827,293	12%	30%
28_District5	Sec	70,112,092,837,500	36,105,072,571,696	37,977,721,729,188	5%	11%
29_District0	Sec	29,353,316,624	39,806,924,541	33,524,358,182	36%	14%
29_District1	Sec	-	7,978,378,039,377	8,674,385,326,423	7%	16%
29_District2	Sec	110,330,052,104	131,489,239,055	134,283,848,007	30%	33%
29_District3	Sec	694,849,542,871	370,499,572,553	311,271,752,652	53%	29%
29_District4	Sec	21,432,228,480,000	9,184,153,773,890	10,573,232,720,568	11%	27%
29_District5	Sec	-	43,202,011,804,846	39,677,743,077,281	21%	11%
3_District0	Sec	90,701,681,116	122,561,494,295	93,891,778,716	35%	4%
3_District1	Sec	32,421,164,072,344	18,208,627,078,860	17,338,725,883,730	5%	0%
3_District2	Sec	189,457,254,091	240,340,957,507	188,867,412,287	30%	2%
3_District3	Sec	1,559,240,037,532	1,281,223,453,464	1,058,243,629,127	53%	26%
3_District4	Sec	185,491,575,000,000	71,112,832,577,682	83,482,029,098,019	6%	25%
3_District5	Sec	127,380,687,115,638	64,891,254,385,918	74,381,254,529,716	6%	21%
30_District0	Sec	14,856,579,555	20,070,374,480	15,051,260,350	35%	1%
30_District1	Sec	6,027,523,726,423	4,312,691,699,679	4,745,923,150,522	0%	10%
30_District2	Sec	28,984,654,012	55,209,612,842	44,675,473,731	90%	54%
30_District3	Sec	155,654,949,956	181,689,072,415	155,105,160,543	30%	11%
30_District4	Sec	14,800,936,453,875	6,693,326,983,565	6,857,770,597,880	10%	13%
30_District5	Sec	-	6,017,528,191,224	5,747,861,271,949	10%	5%
4_District0	Sec	20,379,141,776	25,573,824,722	21,095,649,683	25%	4%
4_District1	Sec	21,349,363,597,639	17,238,235,759,628	16,905,169,777,399	7%	5%
4_District2	Sec	19,771,989,272	29,633,450,013	28,811,661,519	50%	46%
4_District3	Sec	31,243,041,764	49,572,290,641	44,201,463,197	59%	41%
4_District4	Sec	-	51,890,790,231,479	50,049,669,309,215	13%	9%
4_District5	Sec	42,487,073,453,967	31,137,521,929,725	33,227,587,431,189	3%	10%
5_District0	Sec	105,305,055,950	128,781,182,282	105,455,063,383	22%	0%
5_District1	Sec	44,059,812,883,200	22,894,699,266,342	24,243,237,774,422	0%	6%
5_District2	Sec	49,746,631,101	94,695,484,102	88,686,148,630	90%	78%
5_District3	Sec	880,841,982,257	567,433,555,347	471,680,318,418	55%	29%
5_District4	Sec	150,697,387,546,266	72,230,954,757,531	77,486,245,375,909	0%	7%
5_District5	Sec	-	129,423,894,047,924	142,099,626,859,449	6%	17%
6_District0	Sec	174,024,536,017	220,587,523,298	190,729,161,826	27%	10%
6_District1	Sec	37,015,274,340,097	26,507,707,681,695	28,390,614,875,101	8%	15%
6_District2	Sec	263,920,214,771	410,060,413,098	337,739,353,237	55%	28%
6_District3	Sec	461,157,811,721	365,827,522,770	297,438,401,386	34%	9%
6_District4	Sec	81,645,297,259,838	37,068,601,166,035	37,257,536,735,739	9%	9%
6_District5	Sec	111,442,254,660,225	60,410,529,144,935	63,222,970,888,408	12%	17%
7_District0	Sec	72,186,333,969	65,702,903,270	52,906,658,133	24%	0%
7_District1	Sec	-	28,027,021,670,451	28,871,951,159,415	8%	11%
7_District2	Sec	100,049,230,045	116,641,823,753	113,378,156,358	20%	17%
7_District3	Sec	1,065,842,507,210	904,801,723,717	716,691,649,177	44%	14%
7_District4	Sec	73,172,517,460,313	33,193,370,485,671	36,871,805,329,467	3%	14%
7_District5	Sec	114,047,907,432,457	59,405,925,904,448	59,704,483,720,913	14%	14%
8_District0	Sec	47,815,962,595	77,888,791,331	59,951,747,049	63%	25%
8_District1	Sec	25,499,260,800,000	13,654,929,607,347	14,575,820,068,014	0%	7%
8_District2	Sec	73,150,043,167	87,907,337,806	76,073,658,601	20%	4%
8_District3	Sec	682,713,934,325	523,196,098,714	484,219,451,752	42%	32%
8_District4	Sec	69,182,154,223,200	30,291,252,419,996	34,237,082,206,020	1%	15%
8_District5	Sec	79,269,739,560,000	43,918,931,439,070	43,661,182,955,862	9%	8%
9_District0	Sec	100,093,204,504	116,737,451,606	96,428,985,656	21%	0%
9_District1	Sec	22,011,551,047,800	12,399,153,285,006	12,660,173,019,162	2%	4%
9_District2	Sec	89,342,594,099	136,483,997,883	125,096,375,768	53%	40%
9_District3	Sec	2,665,375,409,813	1,459,938,832,250	1,319,059,669,158	60%	45%
9_District4	Sec	101,372,622,941,700	41,362,999,296,926	49,021,510,477,686	8%	28%
9_District5	Sec	53,218,167,455,020	26,860,109,133,590	29,538,757,673,979	6%	16%
C101_100t_20w	Sol	-12,304,901,708	40,805,608,647	262,318,269,537	432%	2232%
C101_25t_5w	Sol	9,512,567	280,350,870	233,291,782	2847%	2352%
C101_50t_10w	Sol	-1,079,154,704	1,573,935,875	7,445,018,952	246%	790%
C102_100t_20w	Sol	729,540,000,000	70,133,116,592	217,573,150,573	1460%	4320%
C102_25t_5w	Sol	-37,546,116	329,912,424	155,194,462	979%	513%

Continued on next page

Table D.1 – *Continued from previous page*

Instance	Set	Solver	GHI	RDCR	GHI ratio	RDCR ratio
C102_50t_10w	Sol	-1,102,530,181	5,524,352,905	6,821,678,900	601%	719%
C103_100t_20w	Sol	492,074,731,530	158,699,272,503	217,694,740,165	2312%	3135%
C103_25t_5w	Sol	-43,052,838	327,409,316	108,135,923	860%	351%
C103_50t_10w	Sol	-1,110,321,792	7,261,913,029	4,367,278,264	754%	493%
C104_100t_20w	Sol	403,070,851,861	276,422,711,493	136,229,440,931	3503%	1777%
C104_25t_5w	Sol	-3,002,994	287,359,701	112,140,769	957%	434%
C104_50t_10w	Sol	-1,086,946,666	12,614,845,759	3,755,625,345	1261%	446%
C105_100t_20w	Sol	-12,158,993,843	47,274,196,433	180,561,154,792	489%	1585%
C105_25t_5w	Sol	1,002,191	288,360,633	306,883,825	28673%	30521%
C105_50t_10w	Sol	-1,102,529,615	2,653,093,049	7,413,851,753	341%	772%
C106_100t_20w	Sol	-4,377,233,762	62,691,808,364	165,775,810,795	1532%	3887%
C106_25t_5w	Sol	9,512,567	283,354,541	233,291,782	2879%	2352%
C106_50t_10w	Sol	-1,086,946,590	2,029,753,005	6,229,506,092	287%	673%
C107_100t_20w	Sol	2,237,260,920	70,886,974,907	150,941,831,566	1126%	2286%
C107_25t_5w	Sol	-2,001,420	244,305,725	191,239,531	12307%	9655%
C107_50t_10w	Sol	-1,125,905,162	2,002,481,693	5,010,097,283	278%	545%
C108_100t_20w	Sol	202,739,168,808	77,696,013,979	158,188,594,756	19868%	40556%
C108_25t_5w	Sol	-8,009,073	212,766,397	190,738,962	2757%	2482%
C108_50t_10w	Sol	-1,125,905,177	1,441,475,838	3,132,285,665	228%	378%
C109_100t_20w	Sol	417,953,467,753	106,780,342,246	150,893,195,121	1985%	2763%
C109_25t_5w	Sol	-8,009,264	285,857,515	112,641,475	1198%	533%
C109_50t_10w	Sol	-1,125,905,761	3,810,168,123	3,755,625,454	438%	434%
C201_100t_20w	Sol	-12,596,720,162	-3,112,699,402	24,609,822,634	75%	295%
C201_25t_5w	Sol	-45,555,917	-20,024,054	-6,507,161	56%	86%
C201_50t_10w	Sol	-1,125,905,241	70,128,361	1,320,704,456	106%	217%
C202_100t_20w	Sol	328,706,407,609	4,474,520,475	39,443,803,257	198%	967%
C202_25t_5w	Sol	-45,555,951	17,022,311	-6,506,835	137%	86%
C202_50t_10w	Sol	-1,125,905,362	261,027,371	-522,044,111	123%	54%
C203_100t_20w	Sol	655,248,510,402	-2,188,612,091	-4,547,459,364	65%	28%
C203_25t_5w	Sol	-45,555,956	19,025,149	32,041,113	142%	170%
C203_50t_10w	Sol	-1,125,905,456	253,235,287	-522,044,257	122%	54%
C204_100t_20w	Sol	729,540,000,000	-2,869,516,410	76,771,932,203	61%	1152%
C204_25t_5w	Sol	-45,555,977	65,582,918	-5,505,920	244%	88%
C204_50t_10w	Sol	-1,125,905,585	307,777,448	1,320,704,411	127%	217%
C205_100t_20w	Sol	-12,596,718,688	27,114,575,490	2,237,262,633	315%	118%
C205_25t_5w	Sol	-45,555,927	-20,023,954	-45,555,832	56%	0%
C205_50t_10w	Sol	-1,125,905,404	-471,398,319	120,775,010	58%	111%
C206_100t_20w	Sol	262,123,724,328	26,117,537,853	9,775,842,231	449%	231%
C206_25t_5w	Sol	-45,555,927	61,577,868	-45,555,811	235%	0%
C206_50t_10w	Sol	-1,125,905,541	740,219,295	1,359,662,729	166%	221%
C207_100t_20w	Sol	217,402,922,647	10,626,970,684	31,905,221,827	257%	570%
C207_25t_5w	Sol	-45,555,933	46,559,137	-4,003,742	202%	91%
C207_50t_10w	Sol	-1,125,905,456	136,358,501	112,983,210	112%	110%
C208_100t_20w	Sol	544,139,570,373	46,860,791,812	-12,596,716,874	472%	0%
C208_25t_5w	Sol	-45,555,933	20,526,829	-45,555,782	145%	0%
C208_50t_10w	Sol	-1,125,905,466	1,277,850,103	1,340,183,737	213%	219%
R101_100t_20w	Sol	6,319,981,503	20,081,267,458	38,976,352,757	218%	517%
R101_25t_5w	Sol	47,893,887	62,467,584	61,354,912	30%	28%
R101_50t_10w	Sol	709,051,142	1,480,434,141	1,634,104,649	109%	130%
R102_100t_20w	Sol	42,318,726,218	20,032,631,932	35,706,932,930	42%	153%
R102_25t_5w	Sol	34,655,177	57,961,950	43,777,594	67%	26%
R102_50t_10w	Sol	641,955,626	1,409,442,931	1,430,220,614	120%	123%
R103_100t_20w	Sol	49,716,801,838	25,571,731,629	29,132,967,105	105%	134%
R103_25t_5w	Sol	30,260,803	49,507,034	39,717,003	64%	31%
R103_50t_10w	Sol	570,531,481	1,470,910,936	1,233,262,573	158%	116%
R104_100t_20w	Sol	61,254,341,207	37,836,109,103	25,020,523,247	128%	51%
R104_25t_5w	Sol	29,982,663	45,390,782	34,766,381	51%	16%
R104_50t_10w	Sol	293,491,459	2,061,785,102	1,033,274,435	603%	252%
R105_100t_20w	Sol	14,550,273,831	24,871,913,269	34,882,822,727	195%	314%
R105_25t_5w	Sol	35,155,830	61,299,290	52,399,483	74%	49%
R105_50t_10w	Sol	388,290,958	1,417,667,263	1,431,952,306	265%	269%
R106_100t_20w	Sol	27,457,727,146	24,893,529,569	34,896,332,922	195%	313%
R106_25t_5w	Sol	30,483,330	58,406,946	43,443,955	92%	43%
R106_50t_10w	Sol	303,880,567	1,532,812,191	1,232,397,082	404%	306%
R107_100t_20w	Sol	68,698,350,493	27,455,025,570	26,655,233,372	107%	101%
R107_25t_5w	Sol	22,028,383	57,572,567	38,604,587	161%	75%
R107_50t_10w	Sol	372,274,723	1,483,464,447	1,096,474,397	298%	195%
R108_100t_20w	Sol	54,674,971,463	40,375,989,134	26,655,233,095	174%	81%
R108_25t_5w	Sol	26,033,407	49,562,611	30,205,211	90%	16%
R108_50t_10w	Sol	293,491,506	2,002,481,237	958,820,167	582%	227%
R109_100t_20w	Sol	37,368,662,167	29,797,658,783	29,119,457,223	298%	289%
R109_25t_5w	Sol	26,255,954	61,466,185	43,165,730	134%	64%
R109_50t_10w	Sol	631,133,985	1,475,672,318	1,299,059,662	134%	106%

Continued on next page

Table D.1 – *Continued from previous page*

Instance	Set	Solver	GHI	RDCR	GHI ratio	RDCR ratio
R110_100t_20w	Sol	81,060,000,000	33,069,780,618	31,591,787,345	294%	276%
R110_25t_5w	Sol	34,488,425	65,804,921	38,604,647	91%	12%
R110_50t_10w	Sol	636,761,298	1,544,499,451	1,362,259,383	143%	114%
R111_100t_20w	Sol	46,452,786,127	29,805,765,370	30,754,167,237	222%	232%
R111_25t_5w	Sol	26,144,635	53,511,884	38,493,349	105%	47%
R111_50t_10w	Sol	496,942,686	1,666,137,378	1,169,197,187	235%	135%
R112_100t_20w	Sol	26,311,468	44,337,120,135	27,492,853,180	168409%	104390%
R112_25t_5w	Sol	427,249,792	78,487,215	42,609,596	84%	0%
R112_50t_10w	Sol	-	1,942,744,176	1,098,205,706	225%	84%
R201_100t_20w	Sol	-1,383,419,123	359,372,643	1,896,810,226	126%	237%
R201_25t_5w	Sol	-5,171,519	-1,722,541	-5,171,408	67%	0%
R201_50t_10w	Sol	-125,097,476	-45,880,209	82,682,553	63%	166%
R202_100t_20w	Sol	56,342,105,263	1,194,290,405	2,734,429,952	271%	491%
R202_25t_5w	Sol	-5,171,550	3,116,949	-5,171,379	160%	0%
R202_50t_10w	Sol	-125,097,690	143,285,322	80,518,341	215%	164%
R203_100t_20w	Sol	81,060,000,000	524,193,762	3,558,539,646	167%	556%
R203_25t_5w	Sol	-5,171,615	5,842,635	-999,685	213%	81%
R203_50t_10w	Sol	-125,097,806	202,589,626	216,873,694	262%	273%
R204_100t_20w	Sol	81,060,000,000	475,557,768	5,193,249,167	150%	643%
R204_25t_5w	Sol	-5,060,450	7,010,513	-610,495	239%	88%
R204_50t_10w	Sol	-125,098,025	31,603,644	149,777,957	125%	220%
R205_100t_20w	Sol	52,216,151,204	1,264,542,313	3,558,539,686	251%	525%
R205_25t_5w	Sol	-5,171,759	1,615,002	-5,171,361	131%	0%
R205_50t_10w	Sol	-125,097,876	82,250,461	219,037,847	166%	275%
R206_100t_20w	Sol	43,975,052,396	2,083,248,306	4,369,139,559	357%	639%
R206_25t_5w	Sol	-5,171,814	1,559,270	-999,659	130%	81%
R206_50t_10w	Sol	-125,097,837	148,912,632	82,682,500	219%	166%
R207_100t_20w	Sol	62,091,960,880	602,552,362	5,206,759,108	182%	808%
R207_25t_5w	Sol	-5,171,795	5,341,774	3,449,995	203%	167%
R207_50t_10w	Sol	-125,097,968	217,307,065	147,613,948	274%	218%
R208_100t_20w	Sol	81,060,000,000	402,603,380	6,868,489,045	146%	885%
R208_25t_5w	Sol	-5,060,561	10,348,010	3,728,196	304%	174%
R208_50t_10w	Sol	-125,098,092	22,945,783	13,422,549	118%	111%
R209_100t_20w	Sol	60,462,655,025	1,248,330,191	3,558,539,754	235%	484%
R209_25t_5w	Sol	-5,060,483	9,402,283	-610,280	286%	88%
R209_50t_10w	Sol	-125,098,000	18,184,429	13,422,591	115%	111%
R210_100t_20w	Sol	29,100,542,983	3,672,023,978	3,558,539,551	541%	528%
R210_25t_5w	Sol	-5,171,642	-2,778,915	-554,744	46%	89%
R210_50t_10w	Sol	-125,097,935	69,696,872	12,557,064	156%	110%
R211_100t_20w	Sol	81,060,000,000	6,230,817,534	1,956,253,653	816%	325%
R211_25t_5w	Sol	-5,060,503	24,364,922	8,066,963	581%	259%
R211_50t_10w	Sol	-122,500,836	159,734,540	149,777,853	230%	222%
RC101_100t_20w	Sol	81,060,000,000	31,372,925,622	39,022,287,441	98%	147%
RC101_25t_5w	Sol	27,034,405	58,017,439	39,160,681	115%	45%
RC101_50t_10w	Sol	190,034,190	1,481,300,065	1,834,093,099	679%	865%
RC102_100t_20w	Sol	31,578,277,390	26,501,220,056	33,267,027,657	67%	110%
RC102_25t_5w	Sol	26,645,148	48,338,693	35,155,744	81%	32%
RC102_50t_10w	Sol	578,756,223	1,267,892,724	1,366,155,547	119%	136%
RC103_100t_20w	Sol	52,216,152,208	28,873,576,160	34,091,137,340	65%	94%
RC103_25t_5w	Sol	21,805,916	48,728,087	34,988,998	123%	60%
RC103_50t_10w	Sol	767,922,883	1,539,738,203	1,498,615,437	101%	95%
RC104_100t_20w	Sol	48,892,692,141	39,635,641,792	25,823,017,852	119%	42%
RC104_25t_5w	Sol	25,810,879	62,634,466	34,432,859	143%	33%
RC104_50t_10w	Sol	498,674,346	1,864,394,369	1,565,710,707	274%	214%
RC105_100t_20w	Sol	33,231,901,421	28,954,635,863	38,211,687,016	165%	250%
RC105_25t_5w	Sol	30,761,388	49,284,340	39,494,502	60%	28%
RC105_50t_10w	Sol	448,893,536	1,403,815,303	1,498,182,440	213%	234%
RC106_100t_20w	Sol	57,147,301,655	31,351,309,184	34,909,843,502	172%	203%
RC106_25t_5w	Sol	22,417,683	62,300,598	39,105,303	178%	74%
RC106_50t_10w	Sol	515,989,391	1,596,877,566	1,563,546,359	209%	203%
RC107_100t_20w	Sol	55,499,082,255	37,909,063,145	34,072,223,636	128%	105%
RC107_25t_5w	Sol	30,761,590	62,467,451	47,170,929	103%	53%
RC107_50t_10w	Sol	575,726,019	1,683,019,699	1,562,247,842	192%	171%
RC108_100t_20w	Sol	62,110,875,294	43,577,858,701	32,437,513,809	194%	119%
RC108_25t_5w	Sol	30,372,206	70,088,018	43,332,682	131%	43%
RC108_50t_10w	Sol	630,268,451	1,728,904,280	1,432,818,410	174%	127%
RC201_100t_20w	Sol	-1,378,013,036	456,644,248	248,592,327	133%	118%
RC201_25t_5w	Sol	-5,171,005	2,504,880	12,016,791	148%	332%
RC201_50t_10w	Sol	-125,096,456	-54,970,611	216,009,701	56%	273%
RC202_100t_20w	Sol	51,392,041,422	1,240,226,022	3,558,542,381	260%	559%
RC202_25t_5w	Sol	-5,171,480	1,782,127	3,450,358	134%	167%
RC202_50t_10w	Sol	-125,096,955	144,585,183	286,135,175	216%	329%
RC203_100t_20w	Sol	81,060,000,000	486,368,151	3,545,030,733	163%	560%

Continued on next page

Table D.1 – *Continued from previous page*

Instance	Set	Solver	GHI	RDCR	GHI ratio	RDCR ratio
RC203_25t_5w	Sol	-5,171,505	1,504,158	3,339,080	129%	165%
RC203_50t_10w	Sol	-125,096,765	138,524,276	149,779,489	211%	220%
RC204_100t_20w	Sol	61,267,851,641	-270,192,915	8,516,710,315	70%	1038%
RC204_25t_5w	Sol	-5,060,413	7,177,386	-721,322	242%	86%
RC204_50t_10w	Sol	-125,097,015	11,692,132	11,259,444	109%	109%
RC205_100t_20w	Sol	39,016,882,589	2,077,845,556	5,193,251,473	375%	786%
RC205_25t_5w	Sol	-5,171,462	2,616,354	3,895,548	151%	175%
RC205_50t_10w	Sol	-125,096,766	217,307,535	354,528,890	274%	383%
RC206_100t_20w	Sol	54,726,310,289	1,278,052,796	1,896,811,249	256%	331%
RC206_25t_5w	Sol	-5,171,398	1,614,724	3,339,383	131%	165%
RC206_50t_10w	Sol	-125,097,138	83,548,298	82,684,447	167%	166%
RC207_100t_20w	Sol	79,446,906,218	3,777,402,945	2,761,451,738	522%	409%
RC207_25t_5w	Sol	-5,059,890	15,187,584	7,789,001	400%	254%
RC207_50t_10w	Sol	-122,499,526	24,244,967	218,173,288	120%	278%
RC208_100t_20w	Sol	61,281,361,512	10,316,241,624	4,382,650,520	1179%	558%
RC208_25t_5w	Sol	-5,060,744	28,036,327	12,127,494	654%	340%
RC208_50t_10w	Sol	-125,096,819	216,874,193	149,779,423	273%	220%
hh_0_P0	HHC	16,590,856,807,985	2,132,214,584,277	702,321,387,236	27788%	9086%
ll1_0_P0	HHC	176,832,488,116	111,689,842,478	72,784,299,458	7699%	4982%
ll1_1_P0	HHC	419,461,597,035	80,332,194,268	120,637,824,265	5509%	8324%
ll1_2_P0	HHC	173,611,127,680	111,689,842,478	71,150,253,394	7699%	4868%
ll1_3_P0	HHC	200,004,491,384	111,689,842,478	71,134,714,208	7699%	4867%
ll1_4_P0	HHC	1,021,093,269,750	111,689,842,478	67,975,583,159	7699%	4646%
ll1_5_P0	HHC	264,587,329,019	110,071,370,621	49,876,891,011	8317%	3714%
ll1_6_P0	HHC	244,823,416,616	85,327,578,094	123,937,028,719	6927%	10107%
ll1_7_P0	HHC	1,002,947,832,000	108,592,963,751	67,851,082,572	7483%	4638%
ll2_0_P0	HHC	33,400,330,208	11,461,556,707	3,226,213,735	13328%	3680%
ll3_0_P0	HHC	27,171,470,455	6,962,257,738	3,227,990,140	14544%	6689%
test150-0-0-0-0_d0_tw0	Mov	-	2,071,227,103,475	102,775,032,769	20053%	900%
test150-0-0-0-0_d0_tw1	Mov	-	-17,864,902,529	39,316,631,920	35%	244%
test150-0-0-0-0_d0_tw2	Mov	-	-12,691,662,644	7,587,429,616	46%	132%
test150-0-0-0-0_d0_tw3	Mov	-	-11,932,921,248	71,045,832,934	52%	388%
test150-0-0-0-0_d0_tw4	Mov	-	140,229,289,294	102,775,033,292	689%	532%
test250-0-0-0-0_d0_tw0	Mov	-	35,170,169,091,553	2,868,482,993,417	63762%	5109%
test250-0-0-0-0_d0_tw1	Mov	-	137,849,382,242	1,841,370,128,095	153%	804%
test250-0-0-0-0_d0_tw2	Mov	-	123,321,148,401	2,354,926,561,517	152%	1089%
test250-0-0-0-0_d0_tw3	Mov	-	121,293,950,821	3,125,261,213,030	166%	1813%
test250-0-0-0-0_d0_tw4	Mov	-	410,507,302,096	2,098,148,344,983	357%	1416%
test50-0-0-0-0_d0_tw0	Mov	-842,599,324	10,189,252,978	-842,598,772	1309%	0%
test50-0-0-0-0_d0_tw1	Mov	-842,599,507	-296,465,074	162,283,233	65%	119%
test50-0-0-0-0_d0_tw2	Mov	-842,599,560	-324,551,724	-830,115,369	61%	1%
test50-0-0-0-0_d0_tw3	Mov	-842,598,590	-327,672,308	-830,113,791	61%	1%
test50-0-0-0-0_d0_tw4	Mov	-842,599,754	-149,791,689	-343,278,151	82%	59%